Computer Science – from A to B

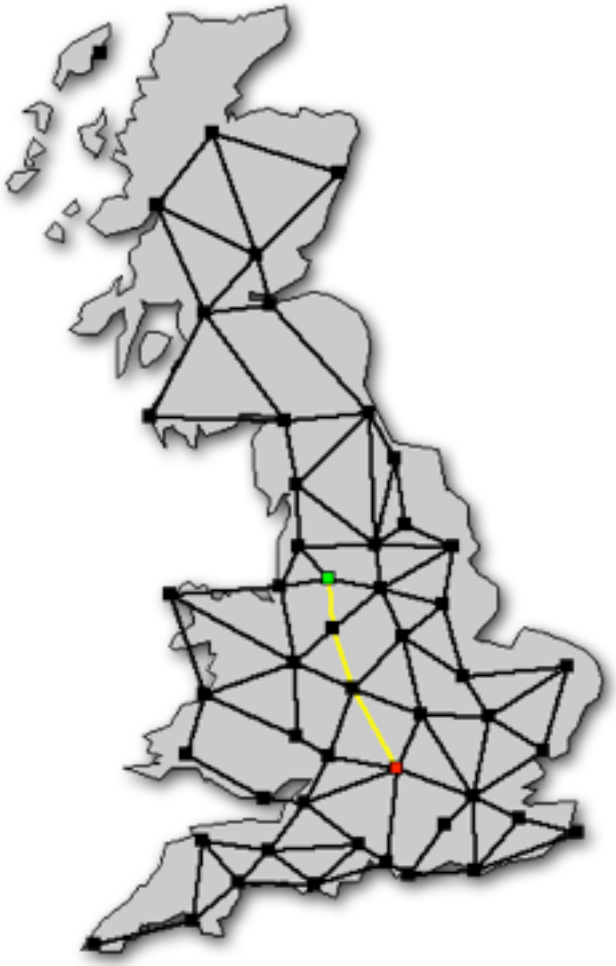
Michael Spivey
Oriel College, Oxford

Department of
COMPUTER
SCIENCE

What is Computer Science about?

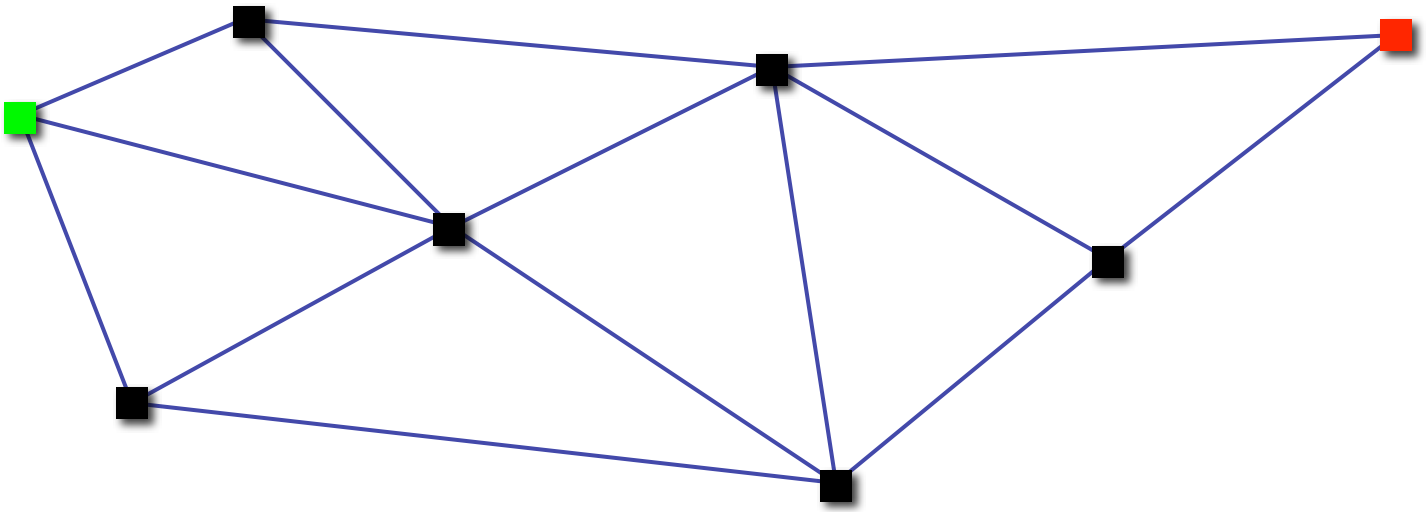
Or, how can you get from Manchester to Oxford?

(And how can we use a computer to answer questions like this?)



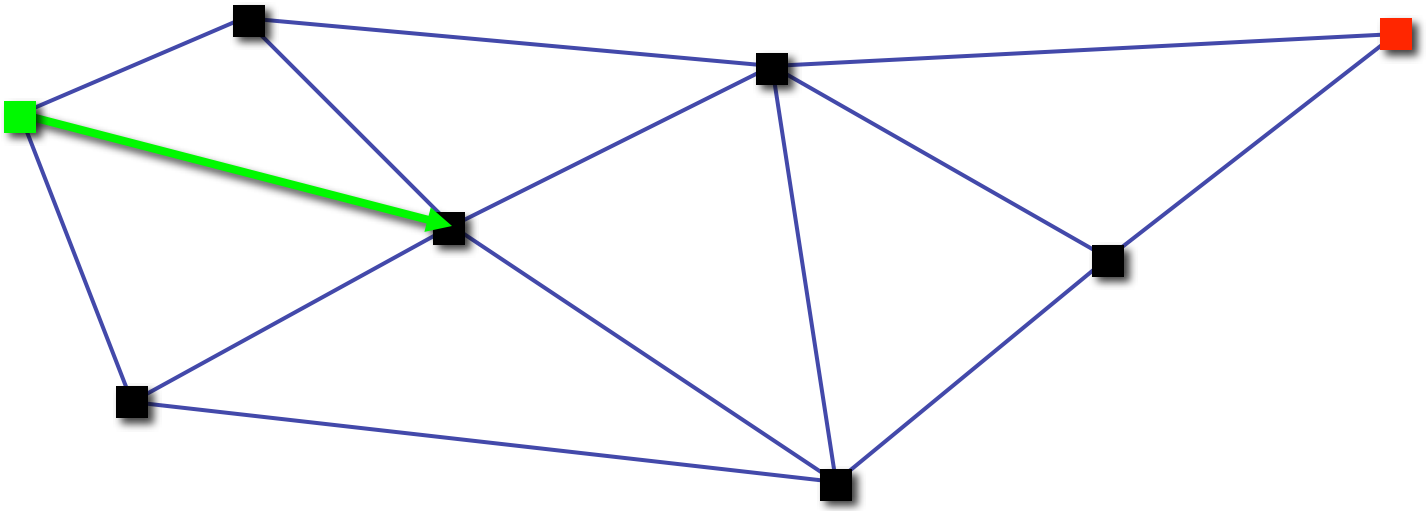
Method A

"Always go in the direction that takes you most directly towards Oxford."



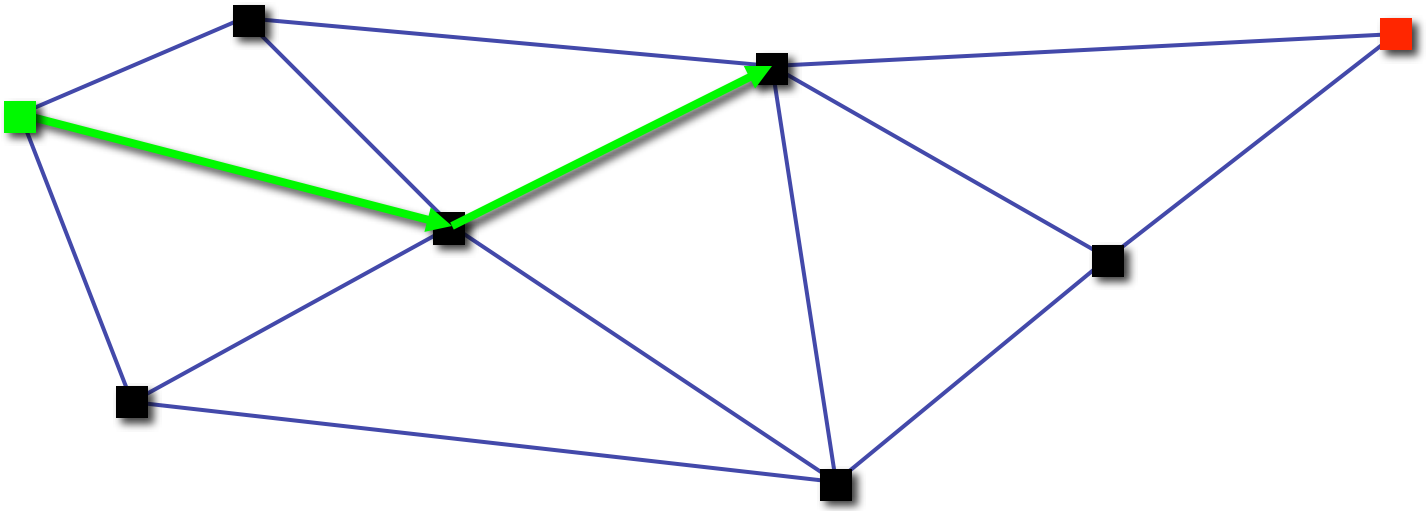
Method A

"Always go in the direction that takes you most directly towards Oxford."



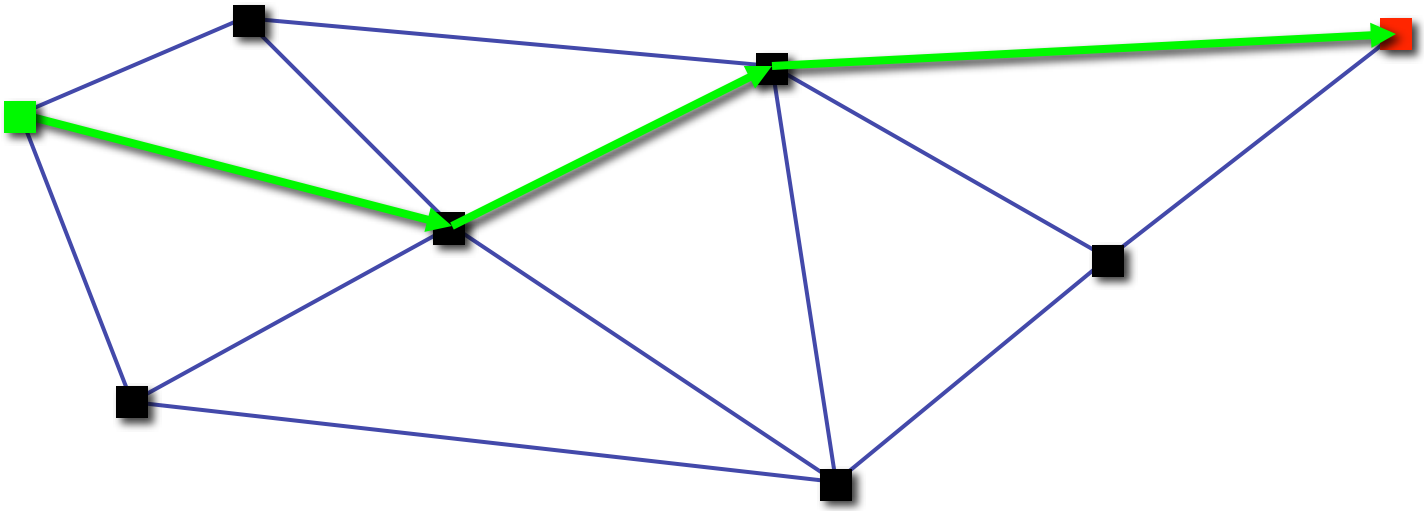
Method A

"Always go in the direction that takes you most directly towards Oxford."



Method A

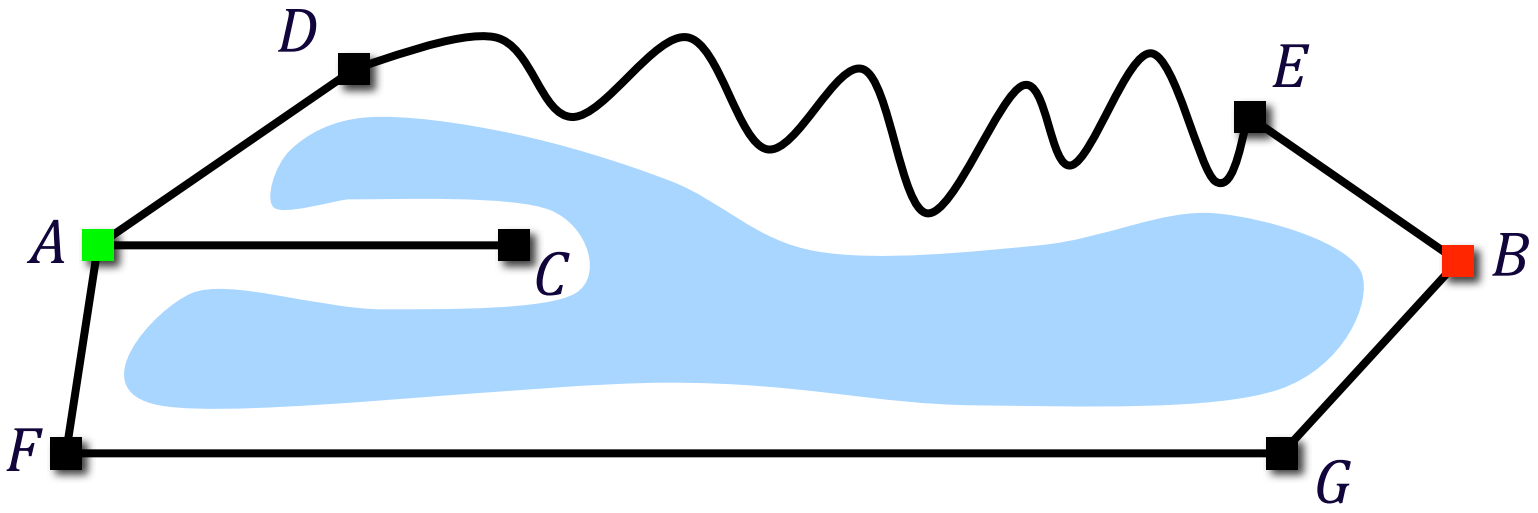
"Always go in the direction that takes you most directly towards Oxford."



How good is it?

Method A

- *almost always* finds a path from start to finish.
- *sometimes* finds a path that not the shortest.



The programmer's dilemma

Any method that …

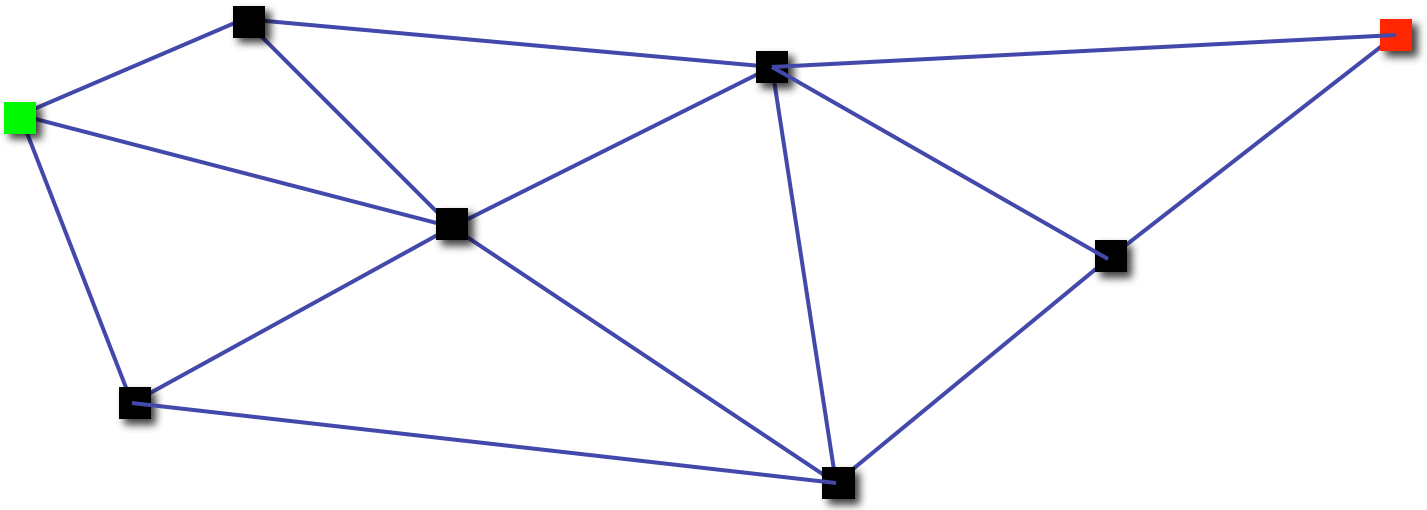
- usually works well but sometimes fails, or
- needs to be used with common sense

… is useless as a computer program.

Method B

Computers are very fast, so

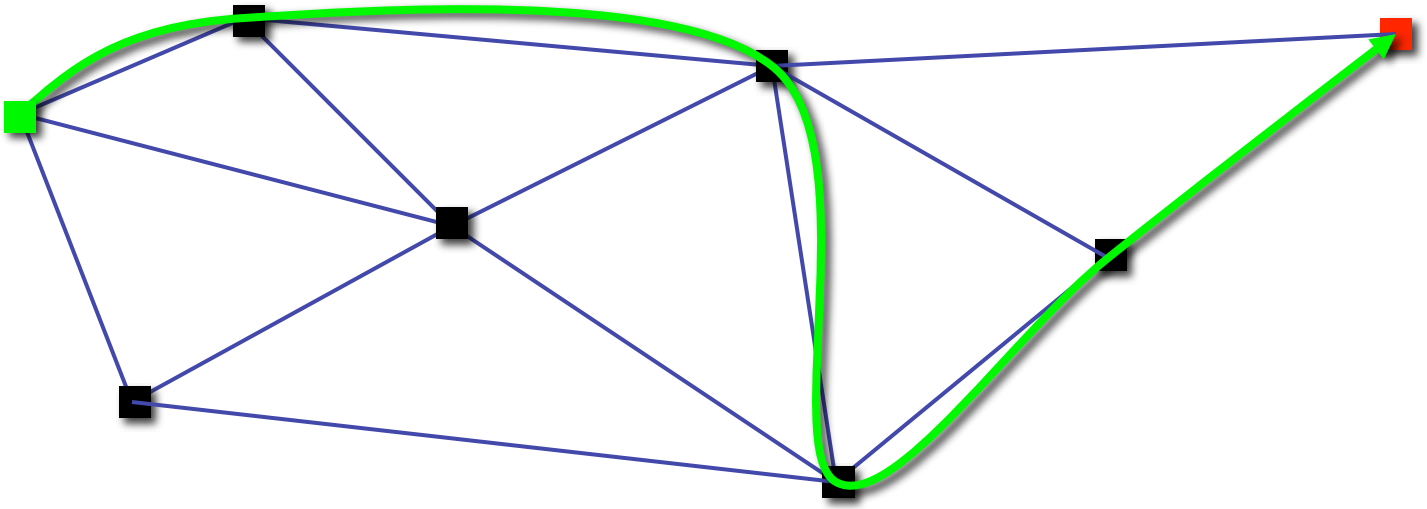
"Consider all routes from Manchester to Oxford, and choose the shortest."



Method B

Computers are very fast, so

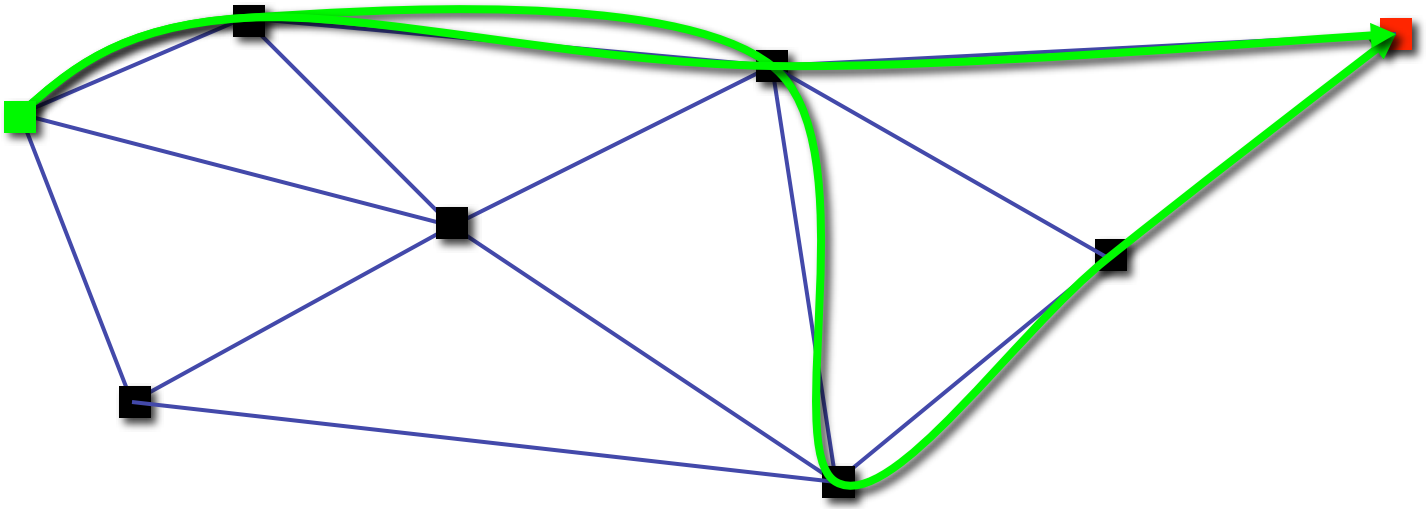
"Consider all routes from Manchester to Oxford, and choose the shortest."



Method B

Computers are very fast, so

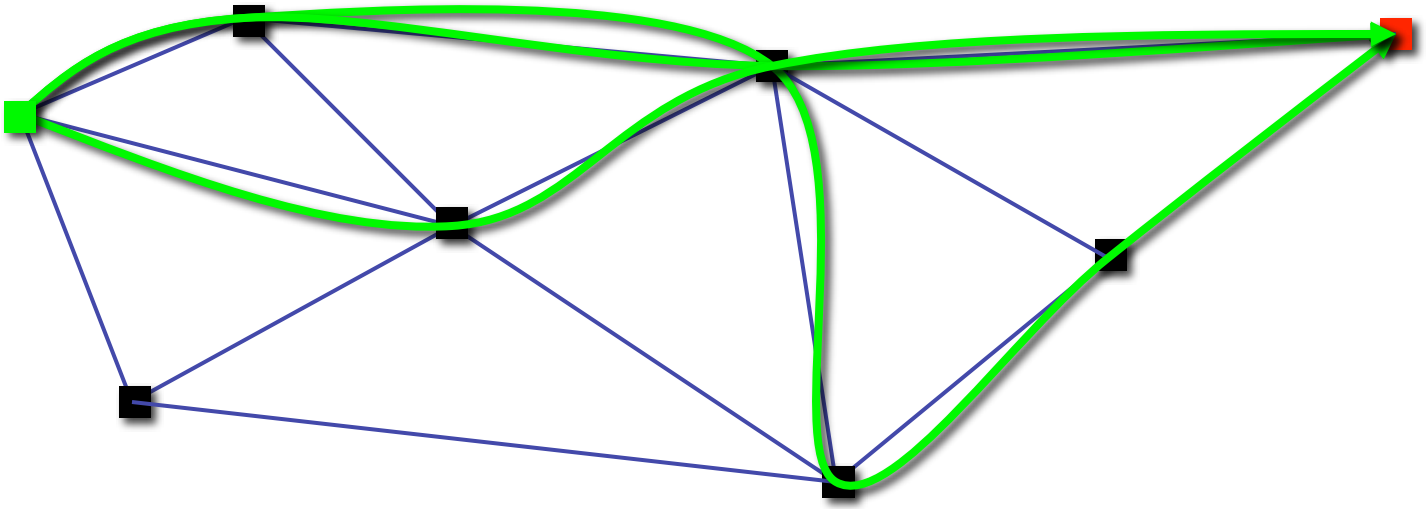
"Consider all routes from Manchester to Oxford, and choose the shortest."



Method B

Computers are very fast, so

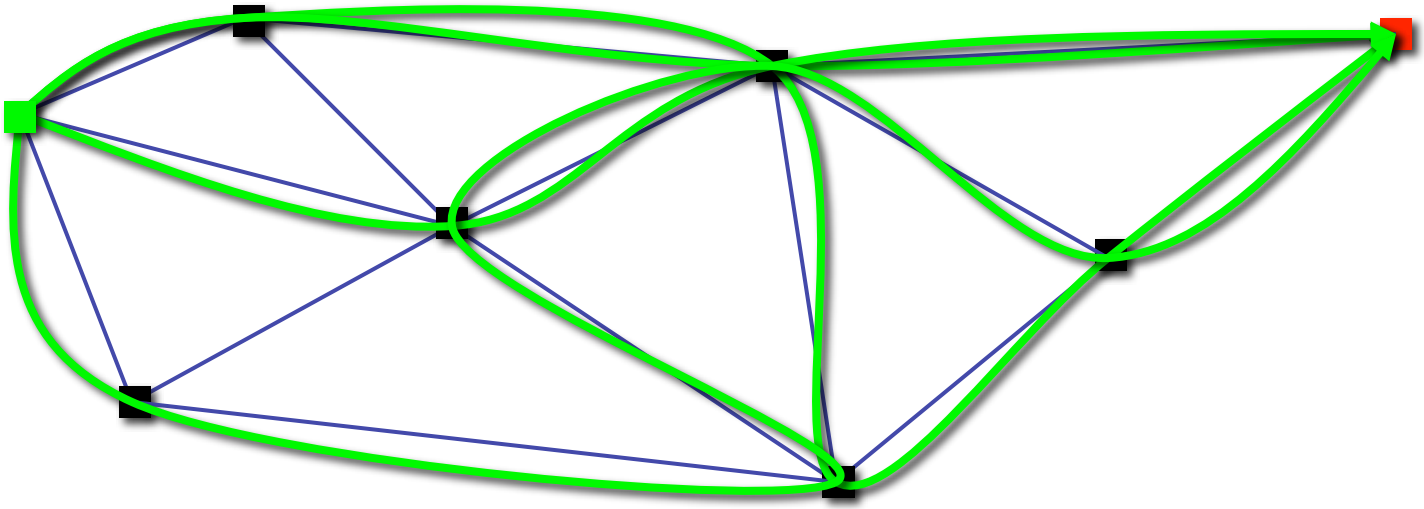
"Consider all routes from Manchester to Oxford, and choose the shortest."



Method B

Computers are very fast, so

"Consider all routes from Manchester to Oxford, and choose the shortest."



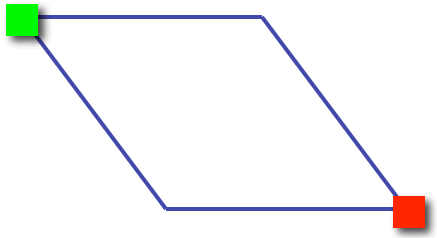
How good is it?

Method B

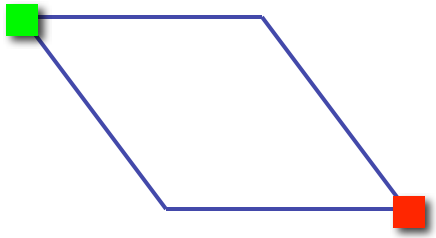
- will certainly find the shortest route.
- but the time taken grows too quickly as the map gets bigger.

How many routes might there be?

Growth of the number of routes

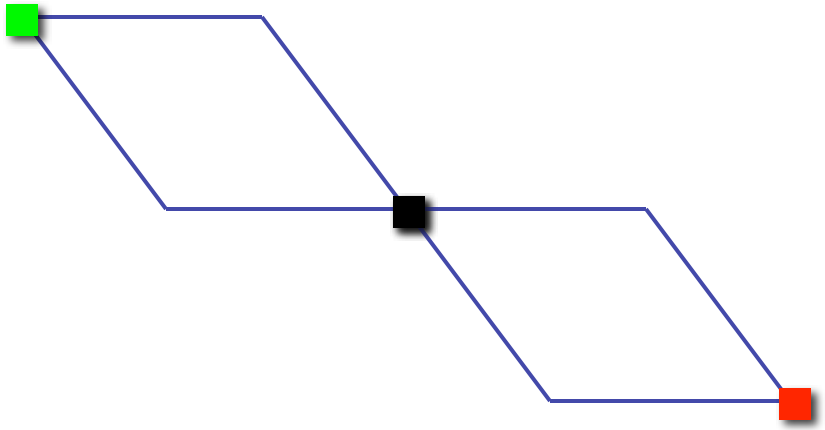


Growth of the number of routes

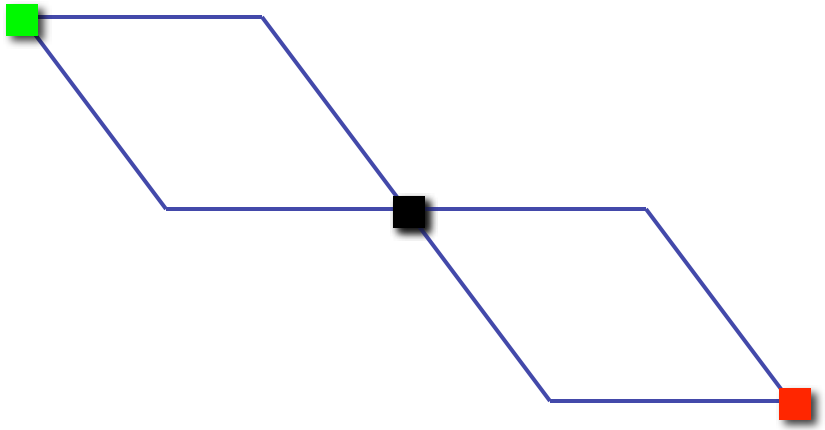


2 routes

Growth of the number of routes

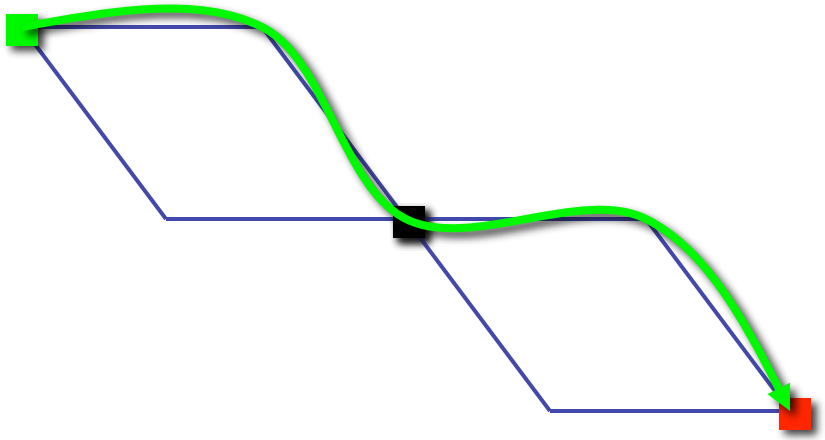


Growth of the number of routes



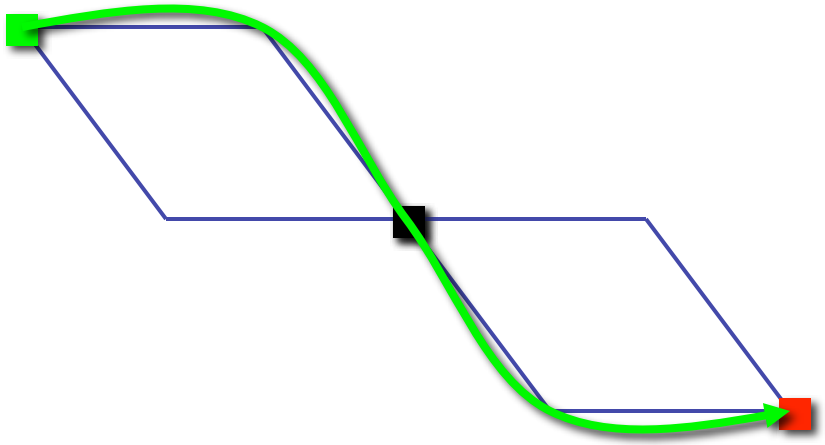
4 routes

Growth of the number of routes



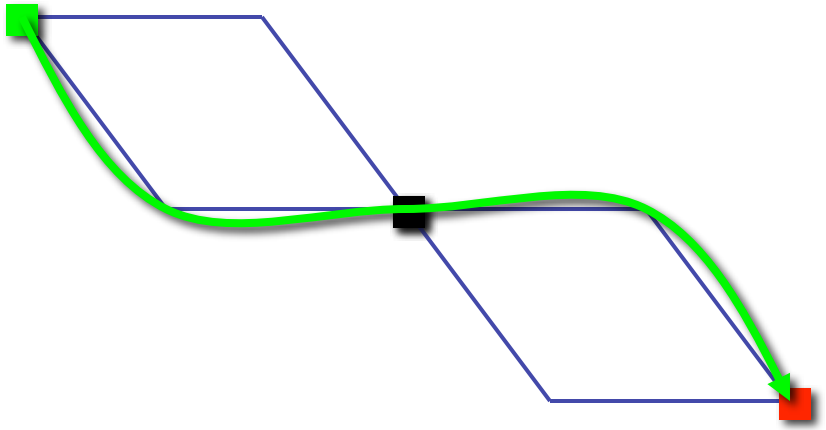
4 routes

Growth of the number of routes



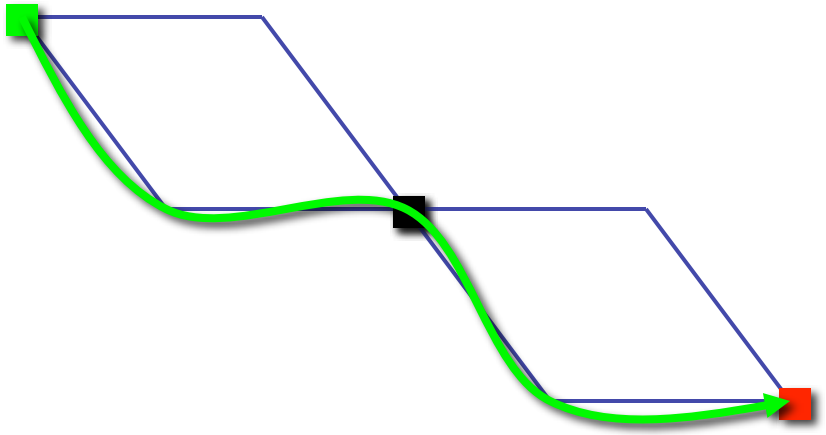
4 routes

Growth of the number of routes



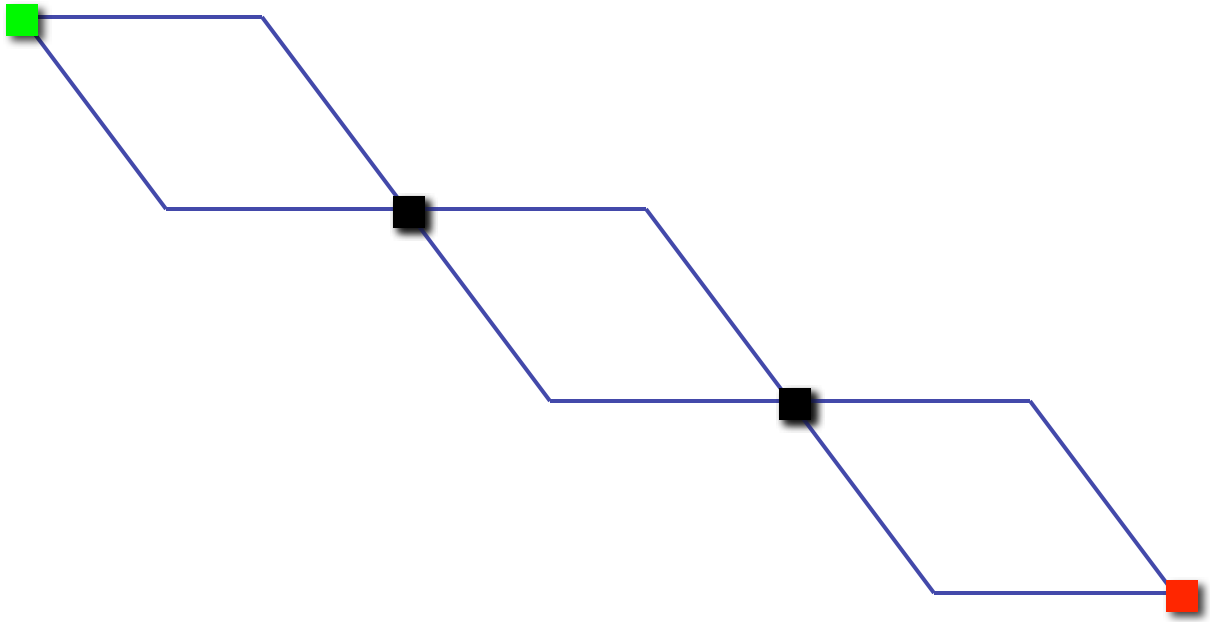
4 routes

Growth of the number of routes

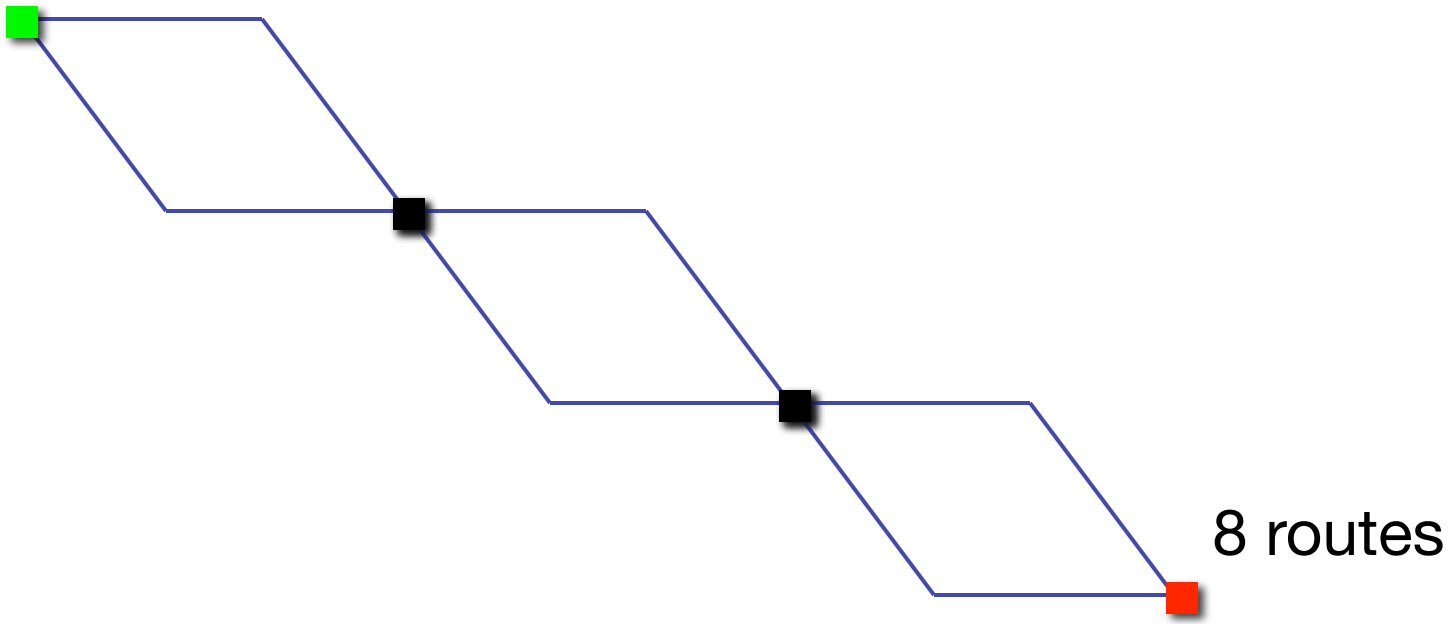


4 routes

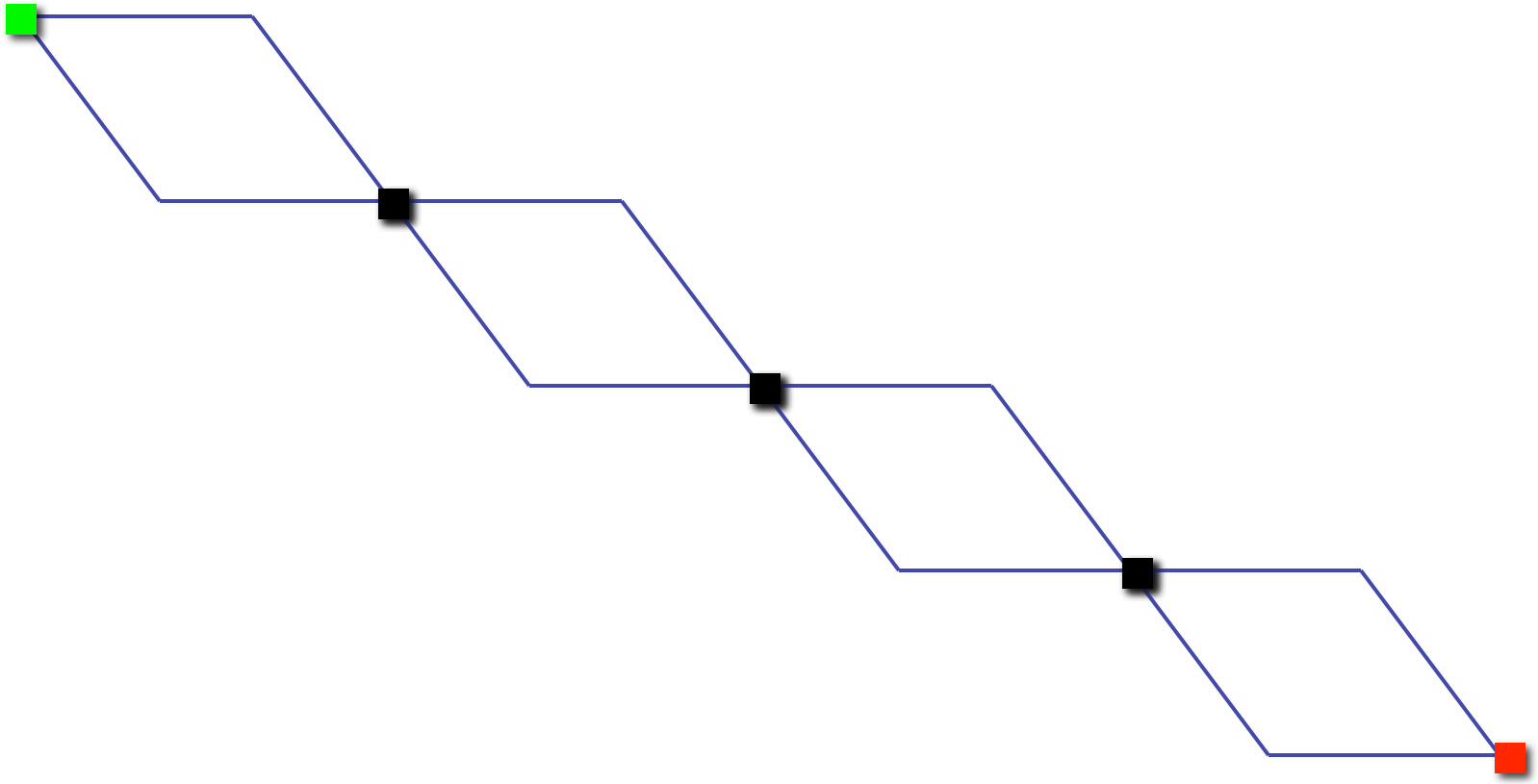
Growth of the number of routes



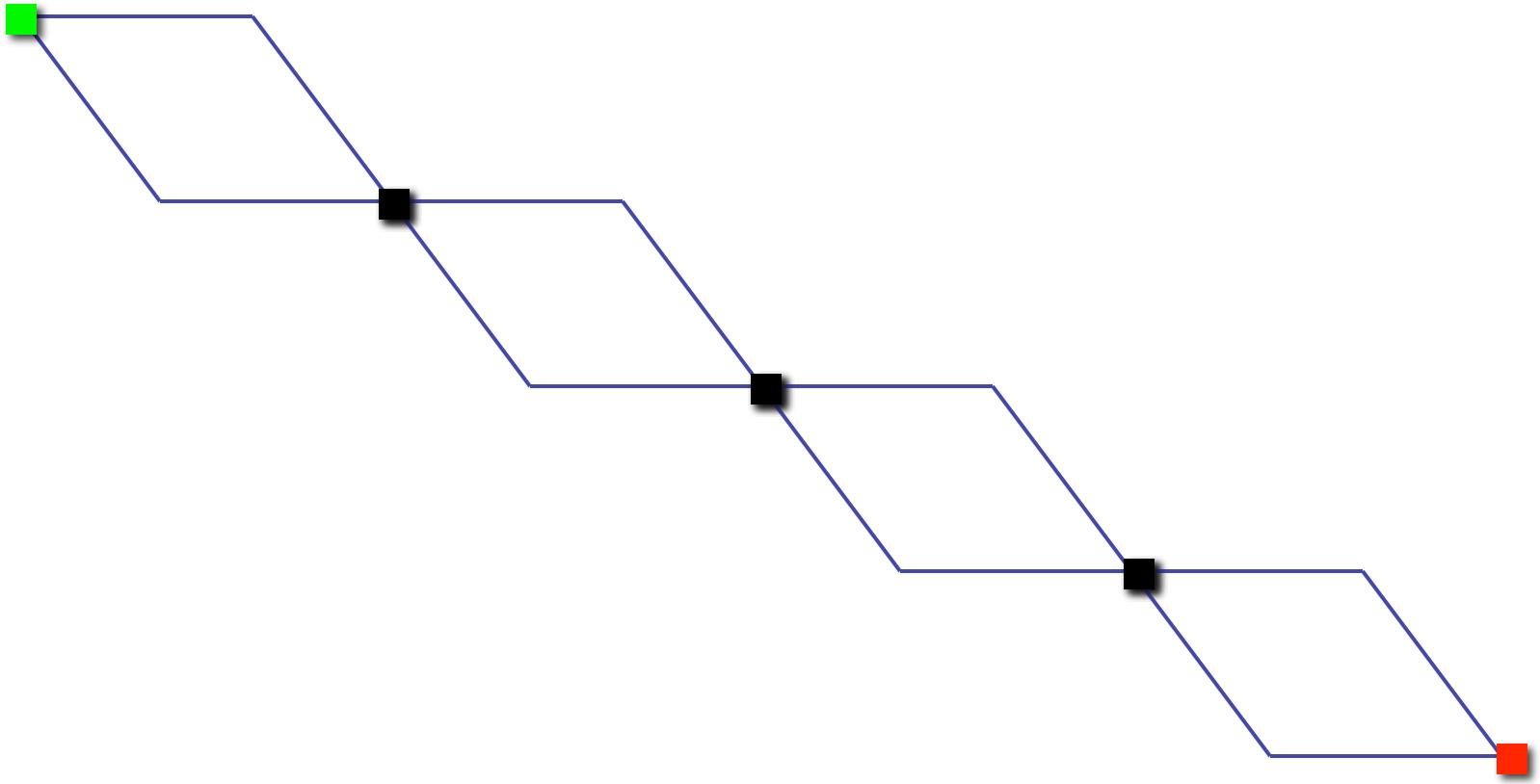
Growth of the number of routes



Growth of the number of routes



Growth of the number of routes



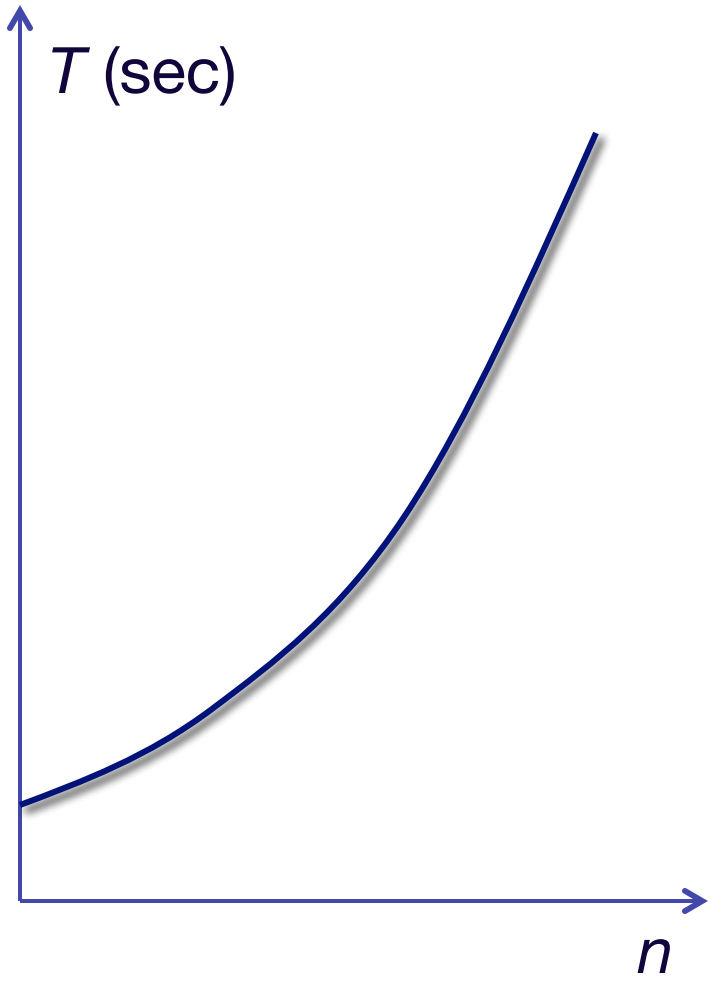
16 routes

Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.

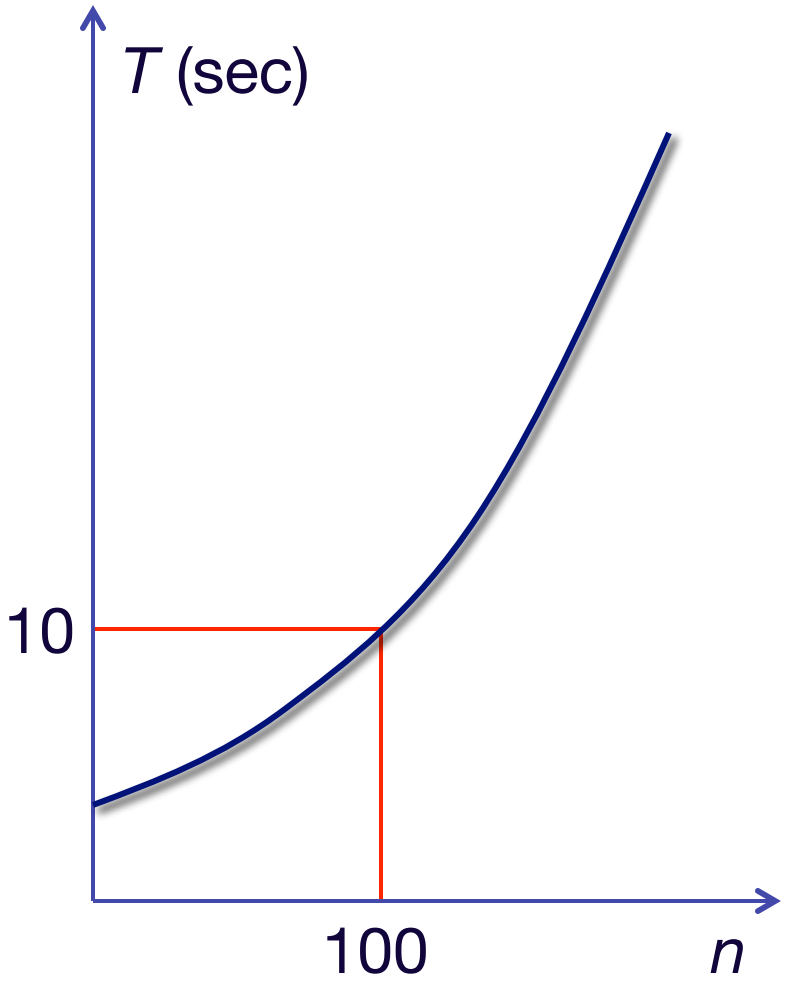
Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.



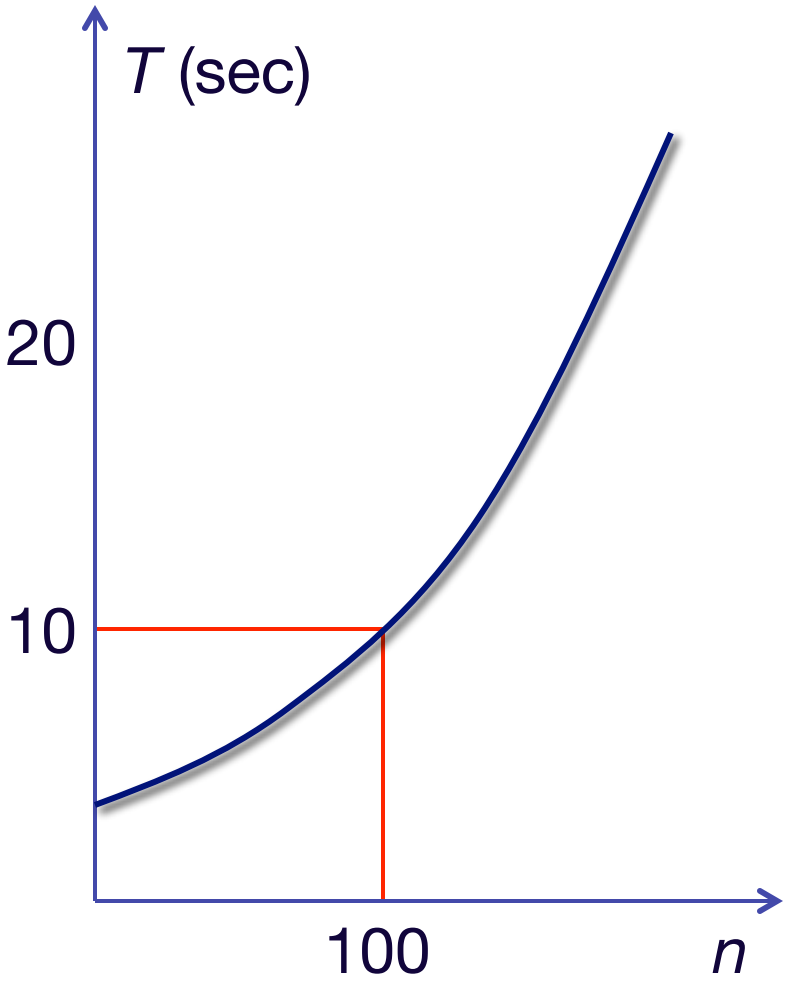
Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.



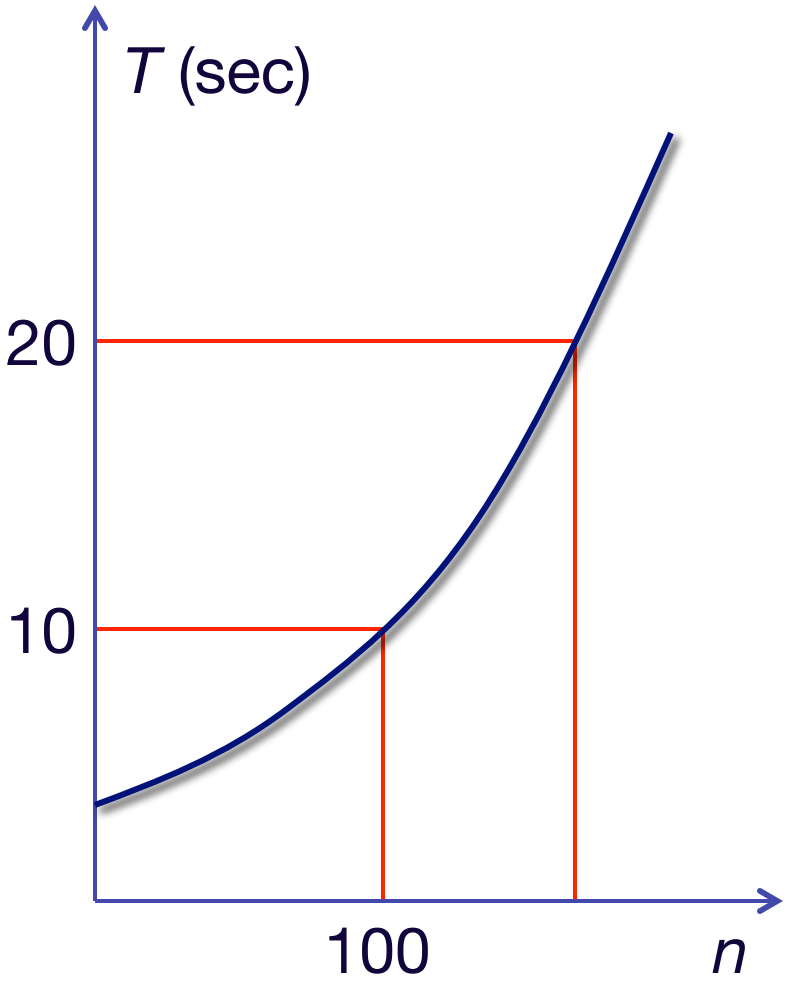
Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.



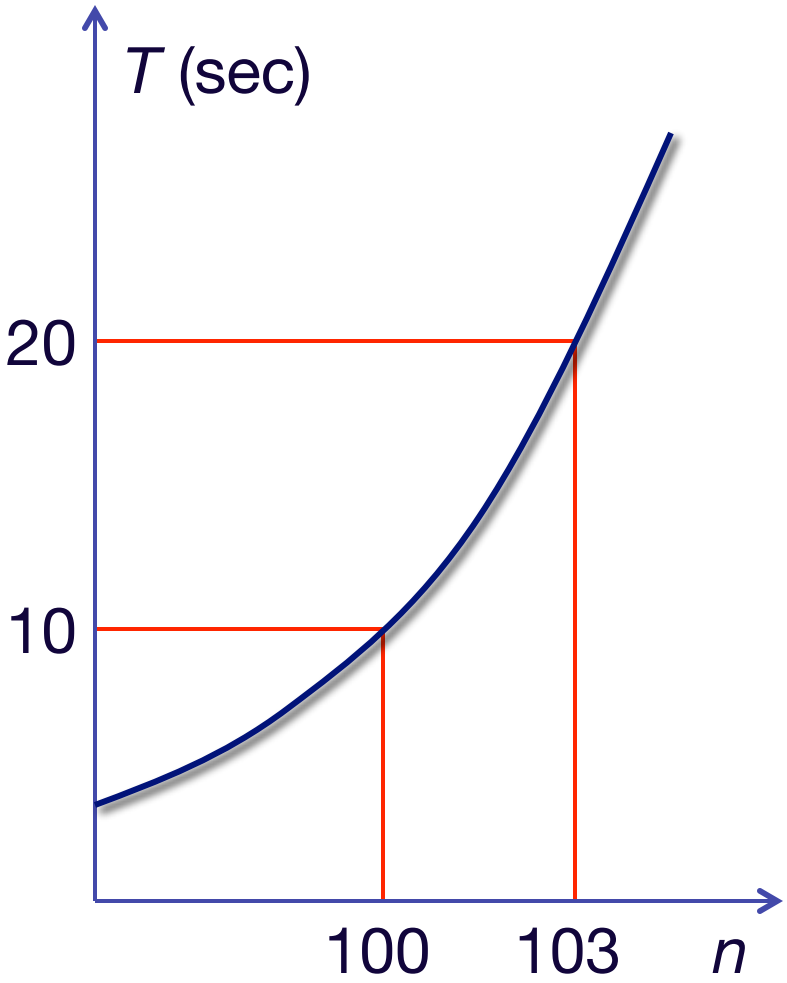
Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.



Method B is slow and gets slower

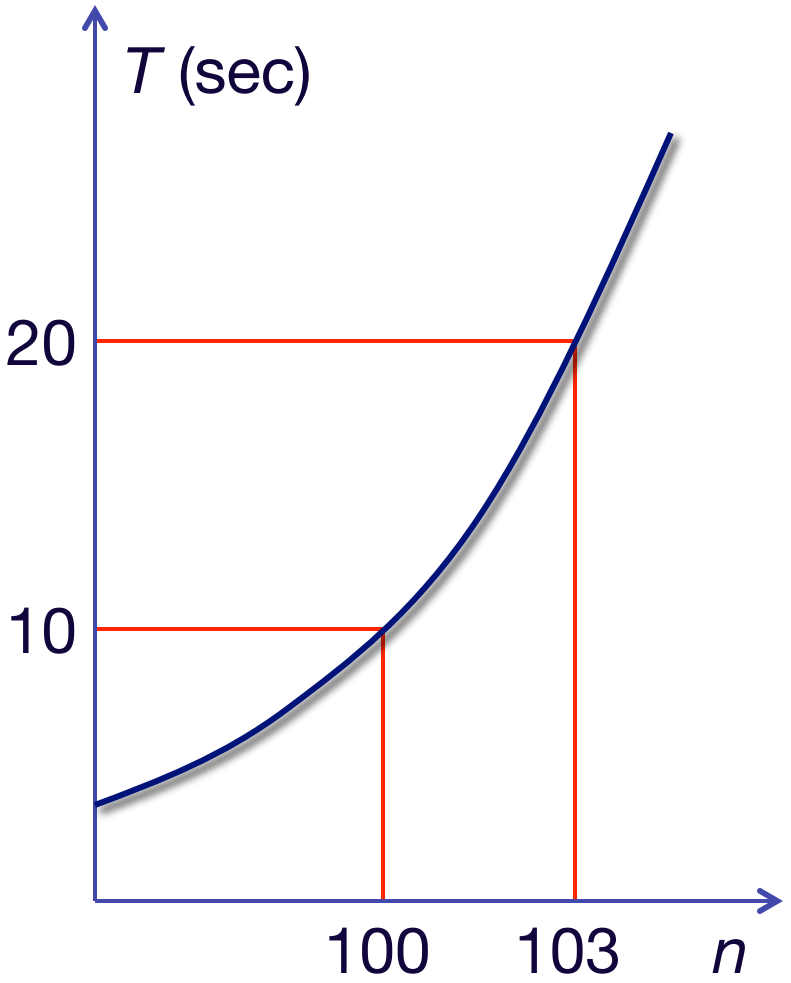
The running time is proportional to $2^{n/3}$, where n is the number of towns.



Method B is slow and gets slower

The running time is proportional to $2^{n/3}$, where n is the number of towns.

- adding three more towns will double the time to find a solution.

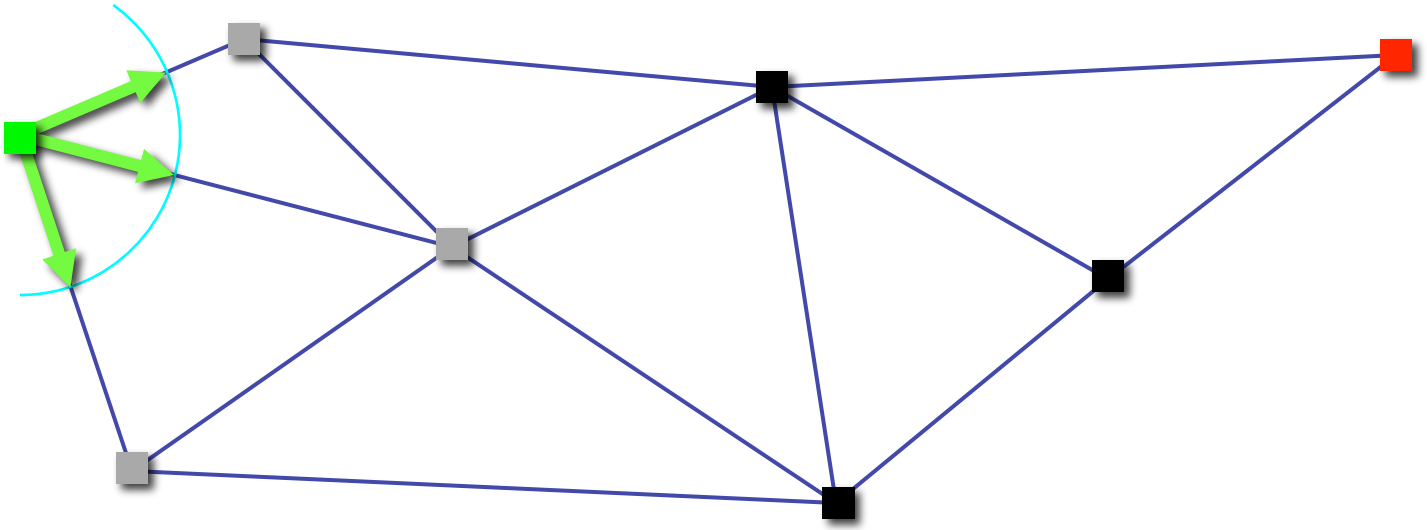


Method C

- Start an army of ants marching on each road out of Manchester.
- Whenever an army captures a new town, send new battalions on each outgoing road.
- The first ants to reach Oxford have gone by the shortest route.

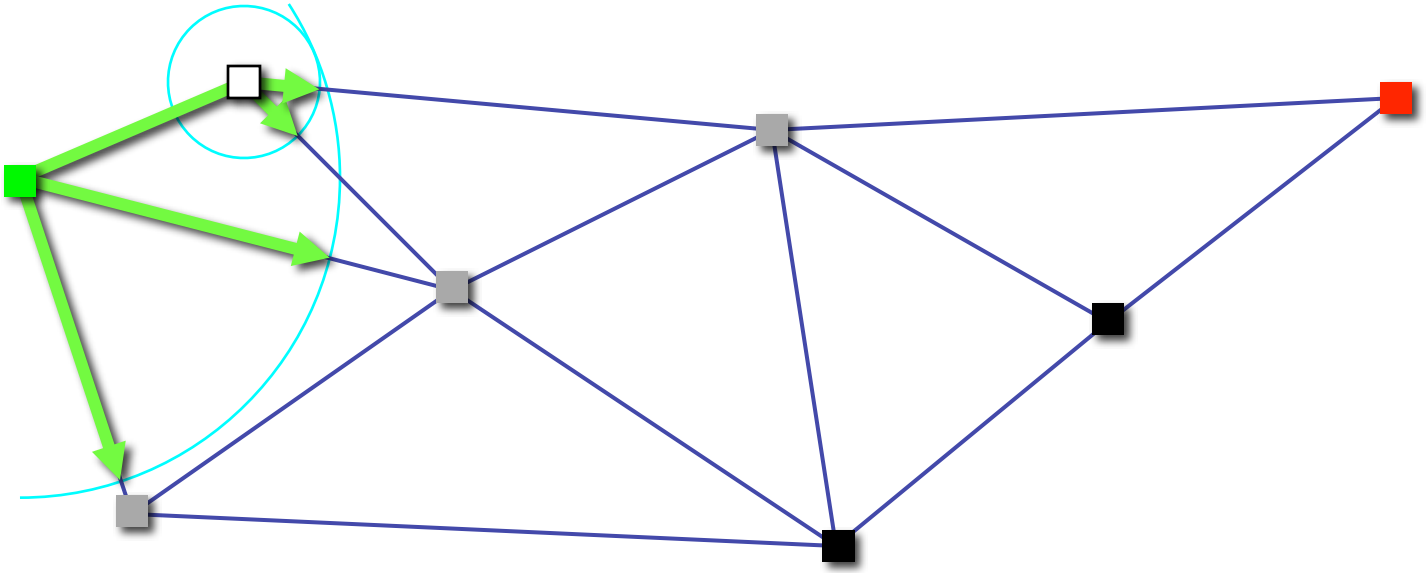
Method C

The ants start out from Manchester, all going at the same speed.



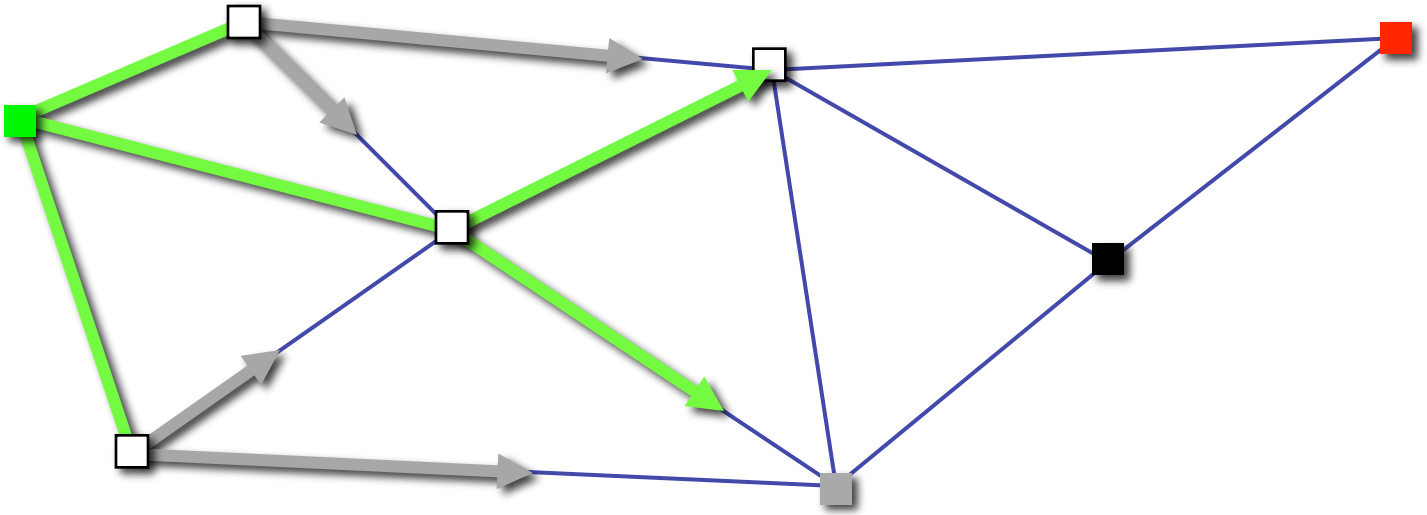
Method C

Each town that is captured becomes a new centre for expansion.



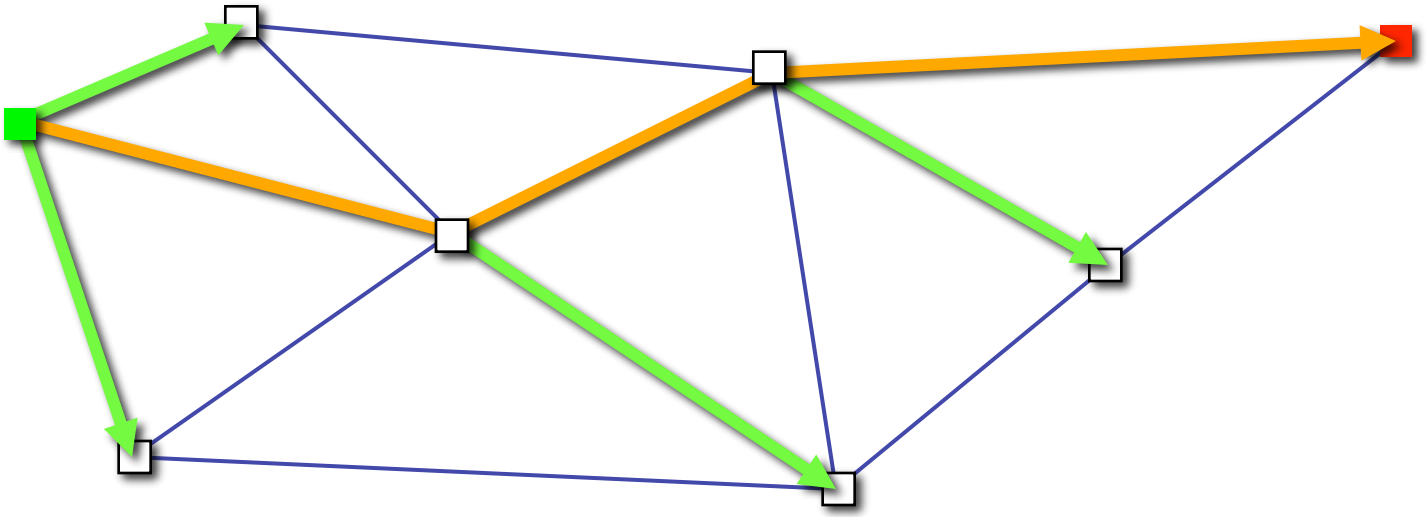
Method C

An army is disbanded if its goal will already be captured before it arrives.



Method C

When the first army reaches Oxford, we have found the shortest route.



Simulating the ants by computer

- **White towns:** already captured.
  - *We already know the best route to get there.*
- **Grey towns:** under attack.
  - *Keep track of which army will get there first.*
- **Black towns:** not yet on front line.

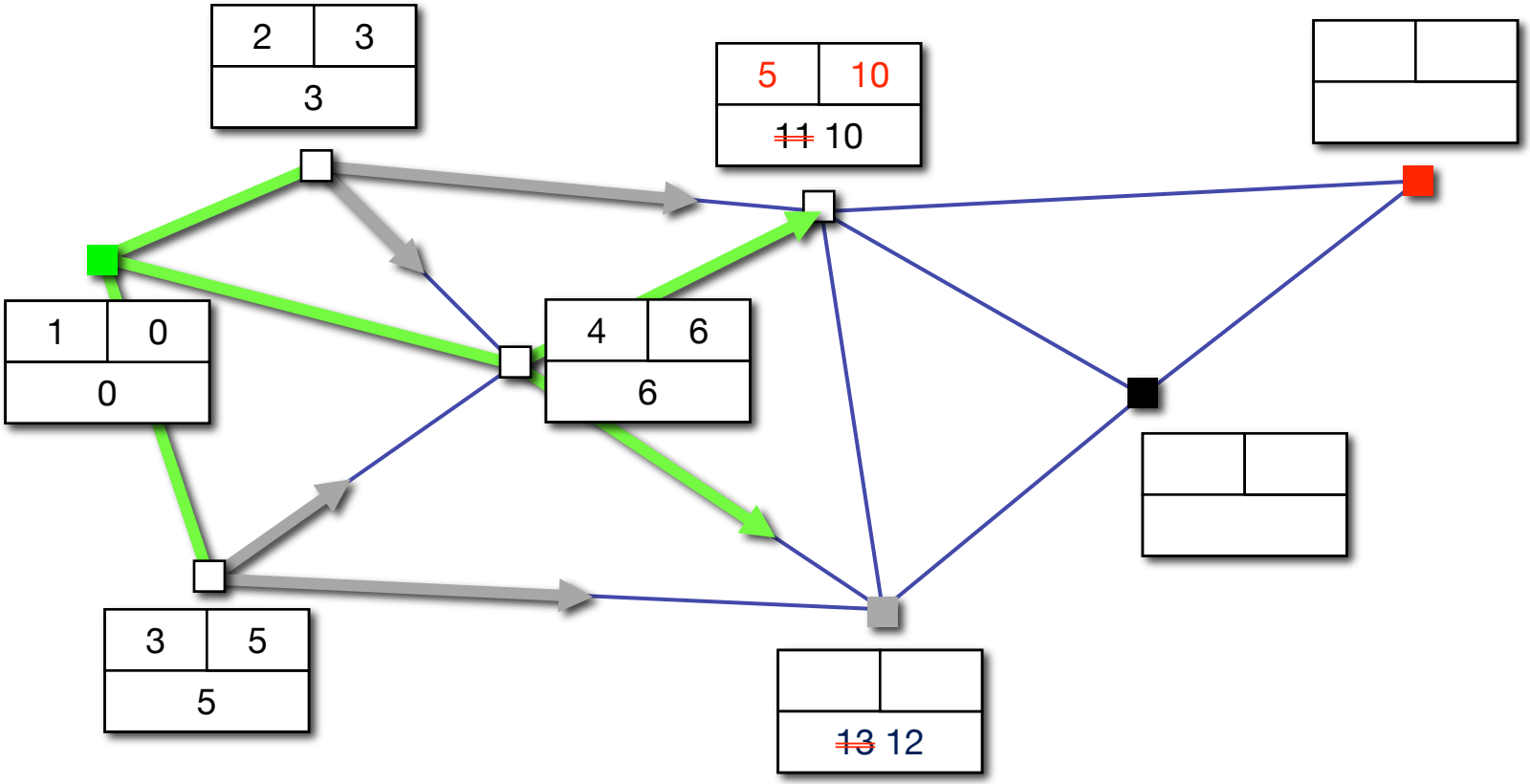
The basic cycle

Repeat these steps …

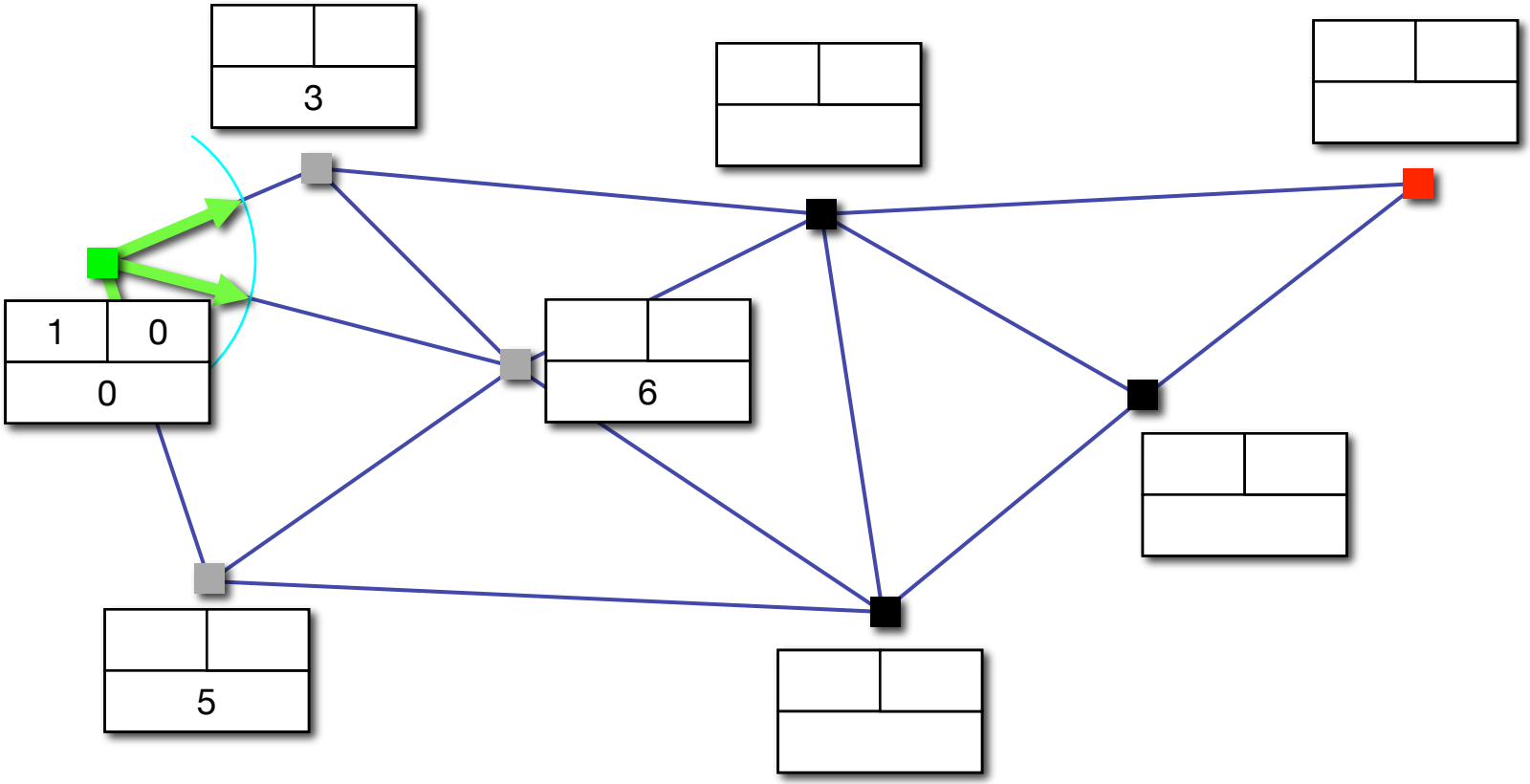
- Identify the next *grey* town to be captured.
- Colour it *white.*
- Colour its neighbours *grey* and update their estimated arrival times.

… until the destination is coloured white.

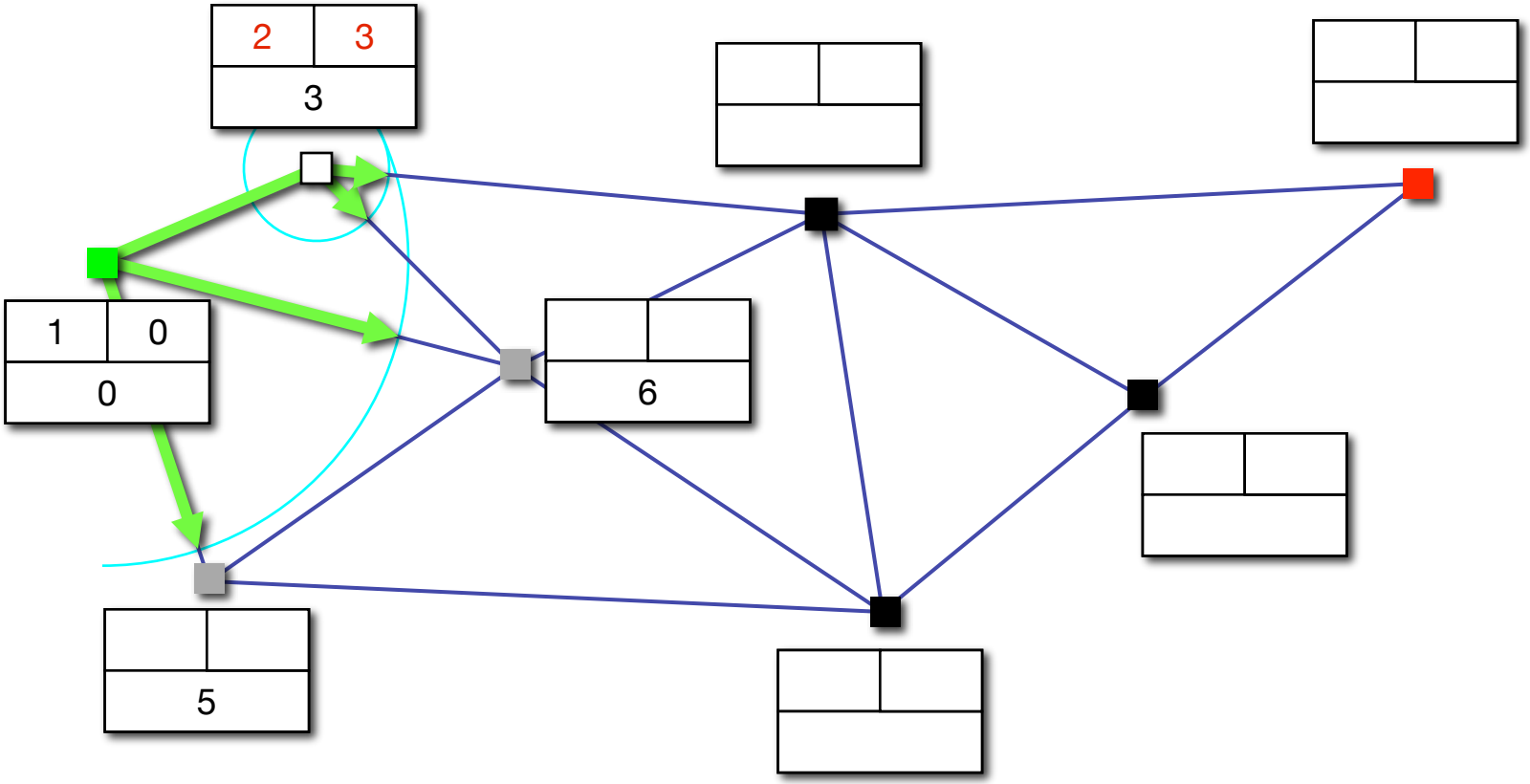
Dijkstra by hand



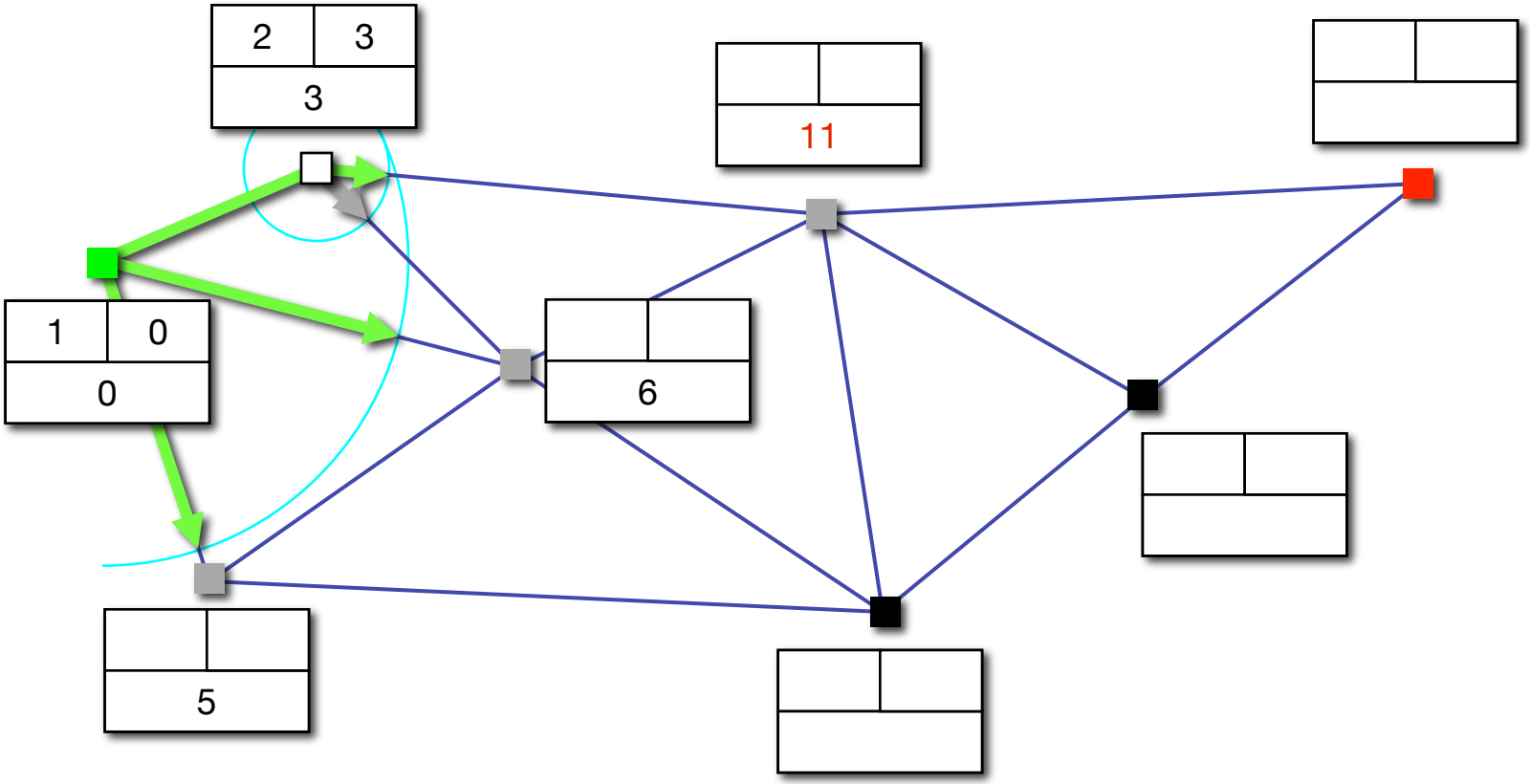
Dijkstra by hand



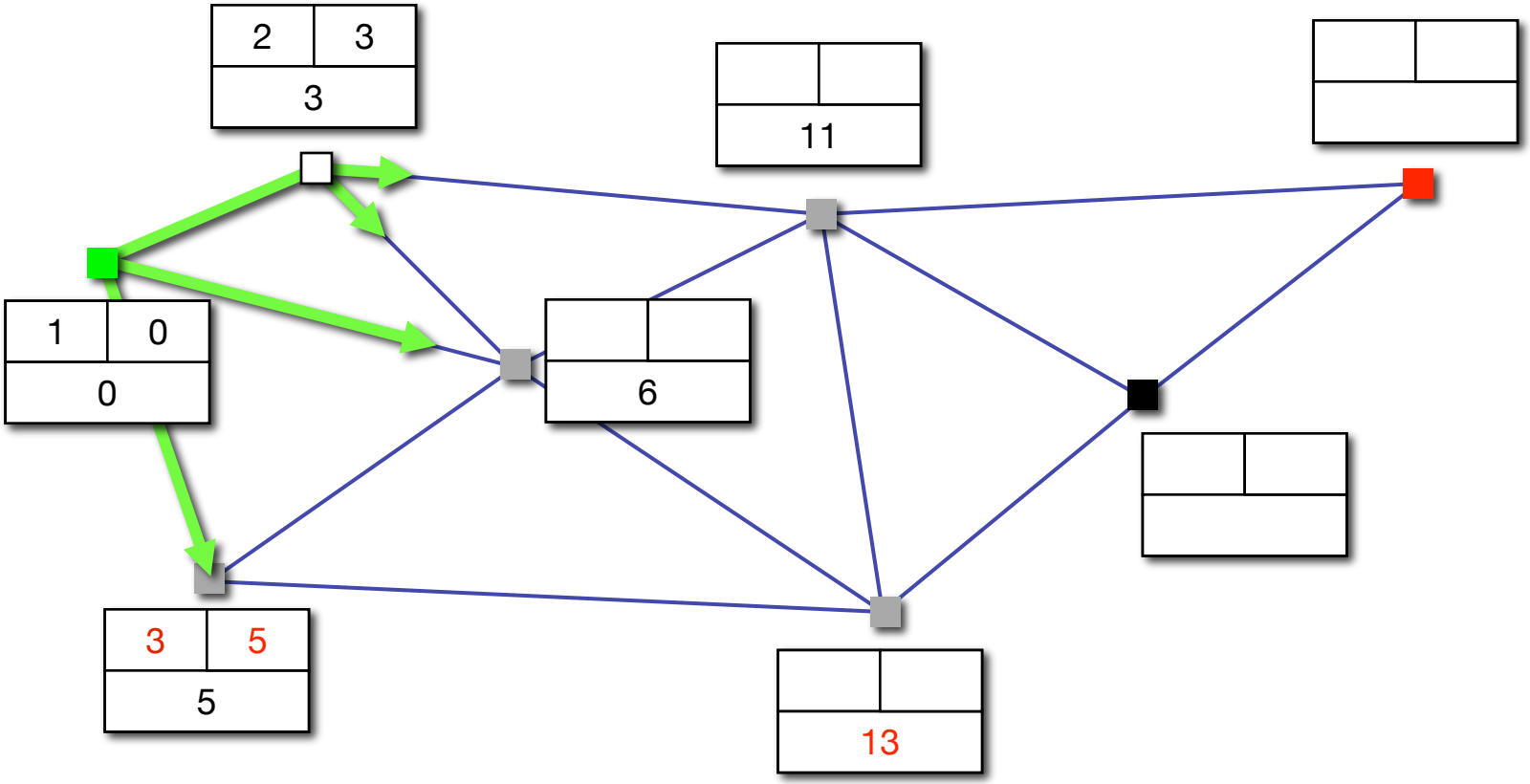
Capturing a town



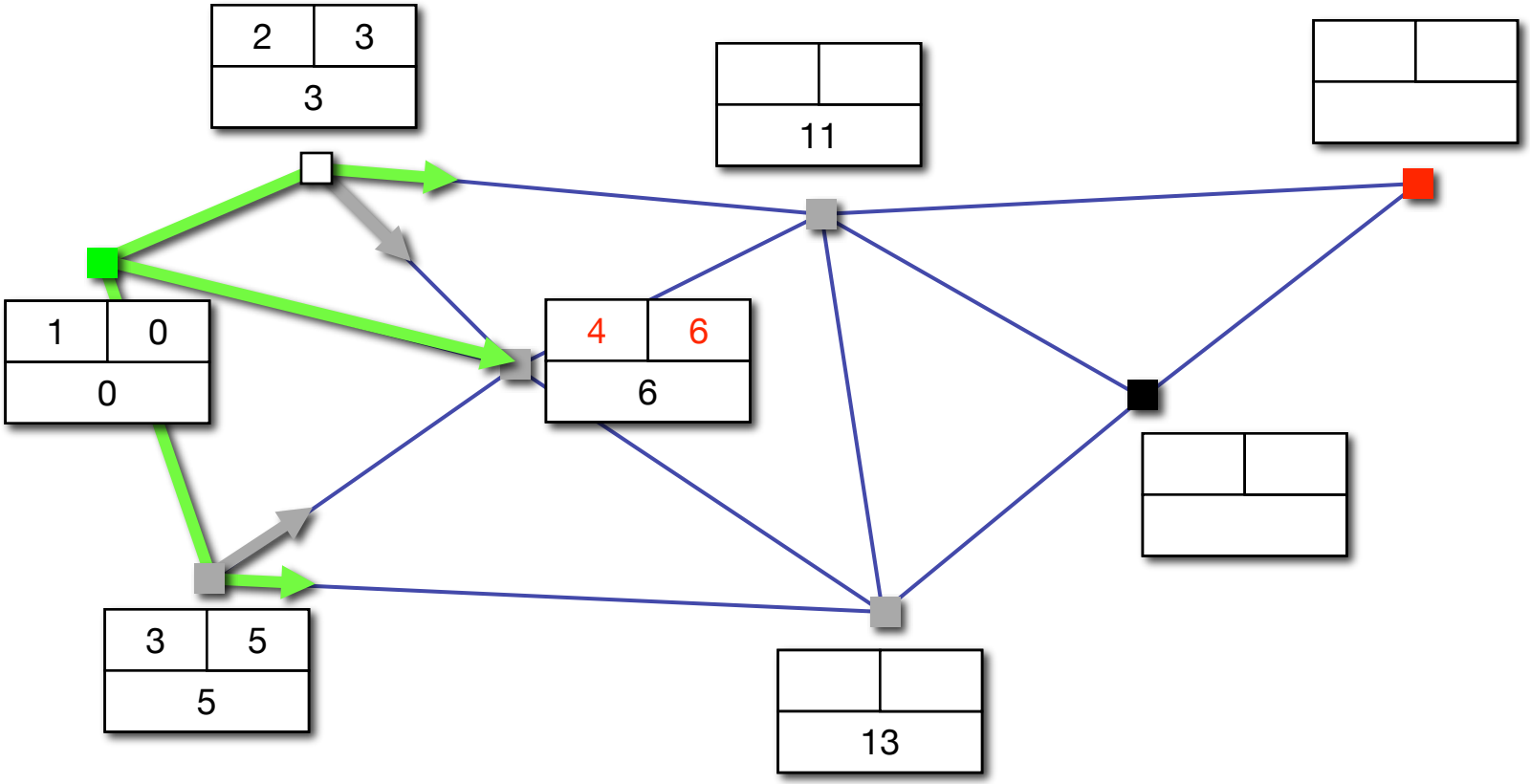
Updating the neighbours



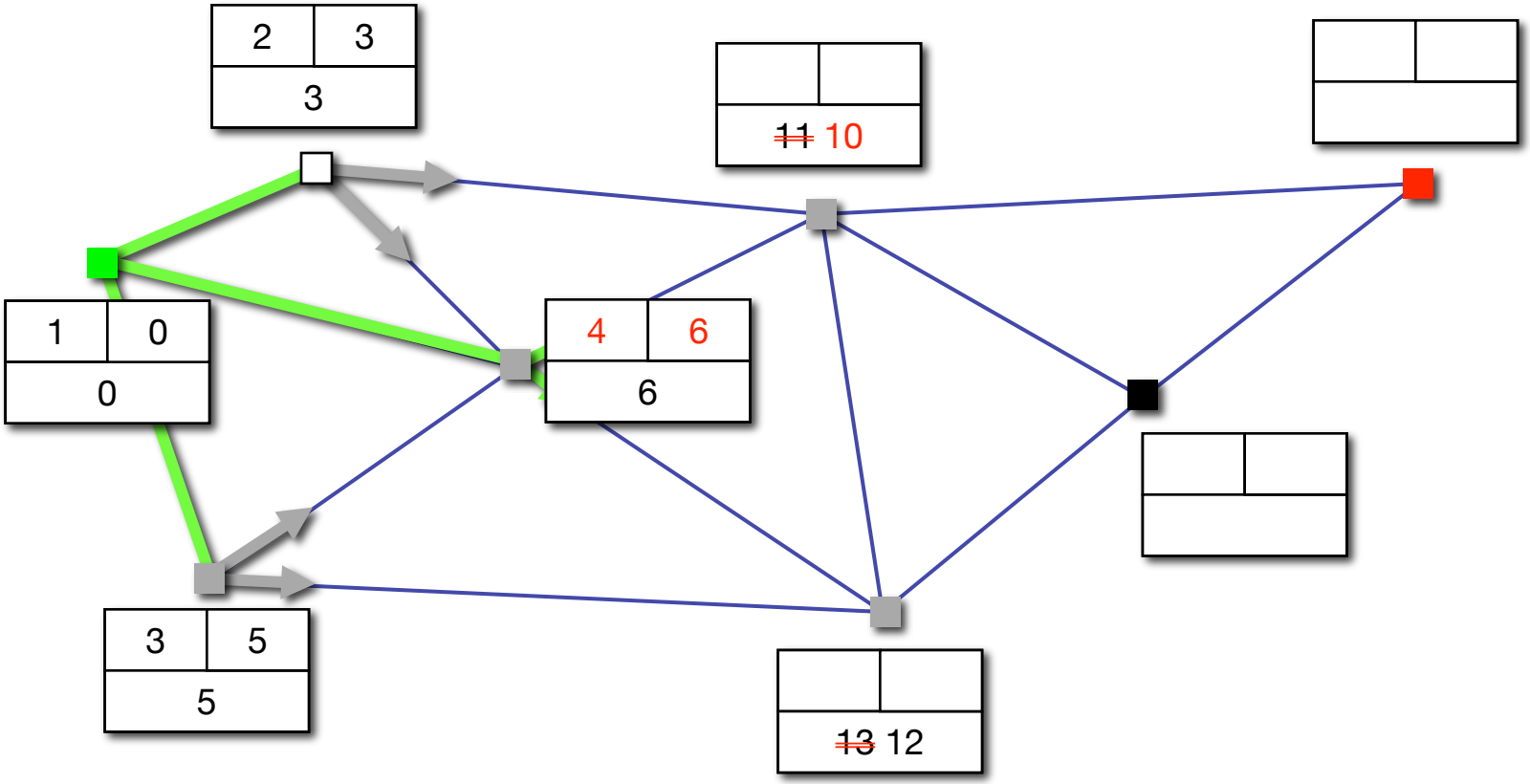
Another capture



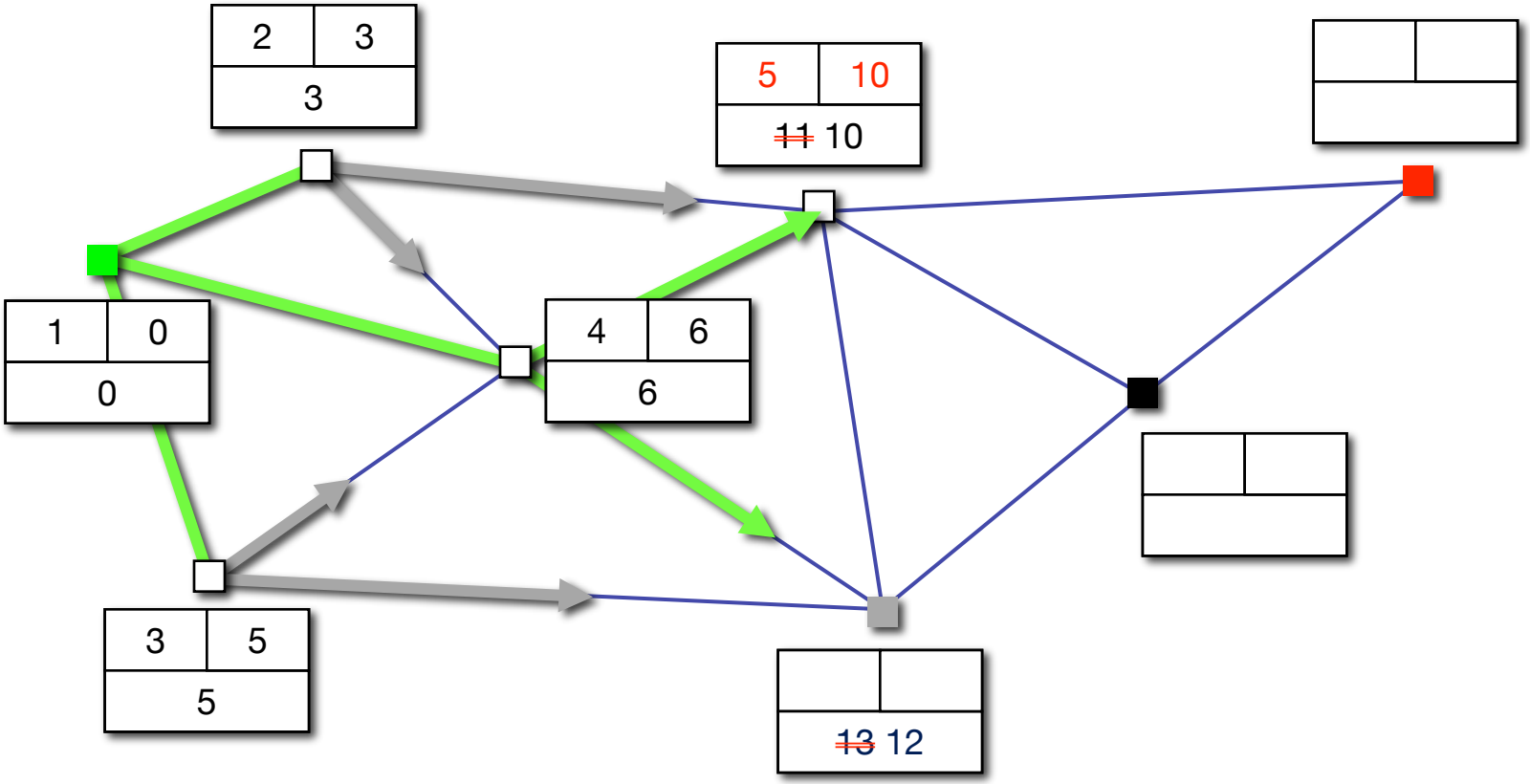
… and another



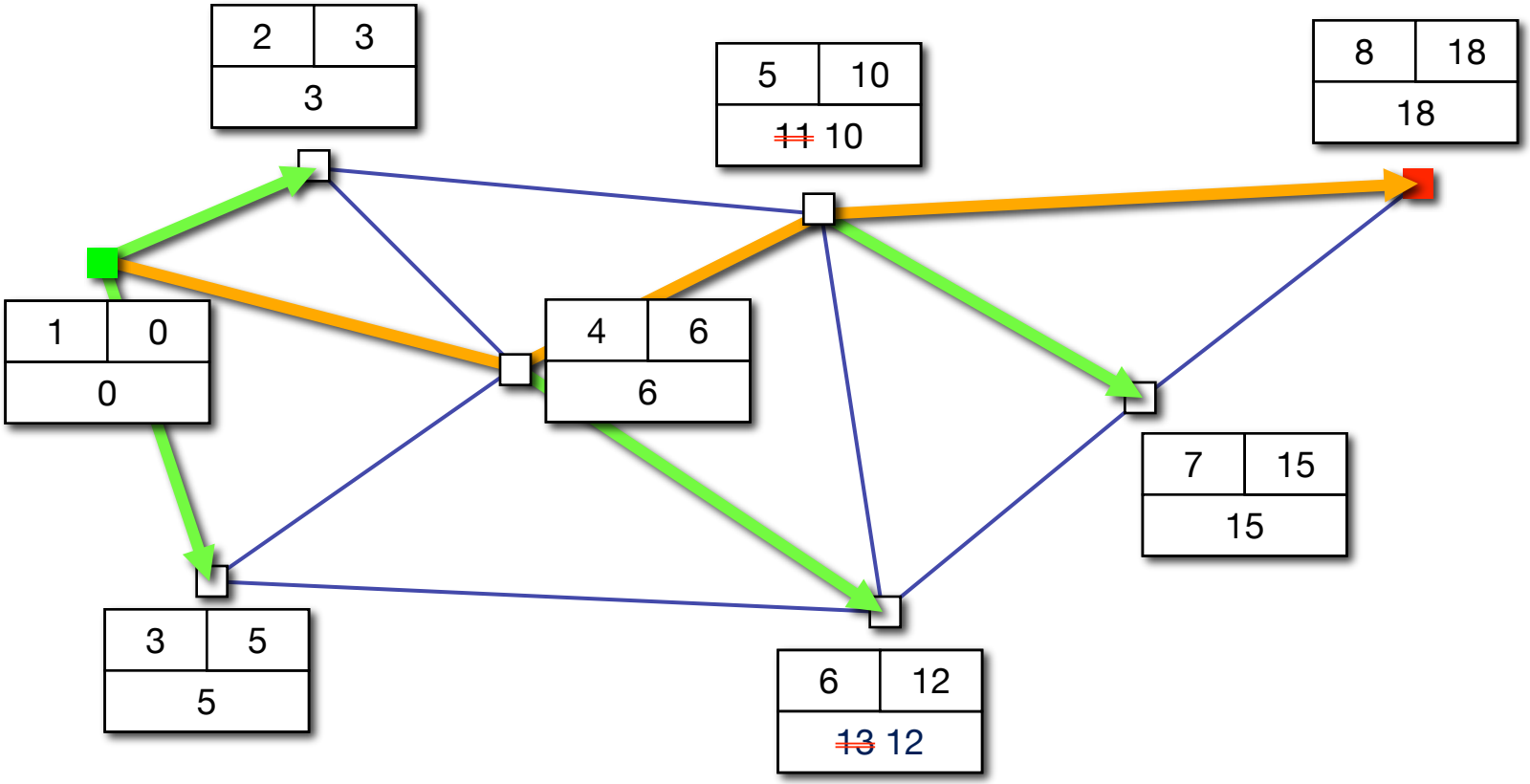
Overtaking



… and getting there first



All finished



Writing it as a computer program

Writing it as a computer program

```
while (dst.colour != WHITE) {
   t = ChooseMin();
   t.colour = WHITE;
   UpdateEstimates(t);
}
```

Java

Writing it as a computer program

```
while (dst.colour != WHITE) {
    t = ChooseMin();
    t.colour = WHITE;
    UpdateE
}
```

Java

```
while dst.colour ≠ white do
    t := ChooseMin( );
    t.colour := white;
    UpdateEstimates(t)
end
```

Oberon Pascal

Why the method is useful

Method C (Dijkstra's algorithm)

- Always gives the right answer.
- Does so in a predictable time.
- The time taken grows only gradually as the map gets bigger.

[Demo]

Guiding the search

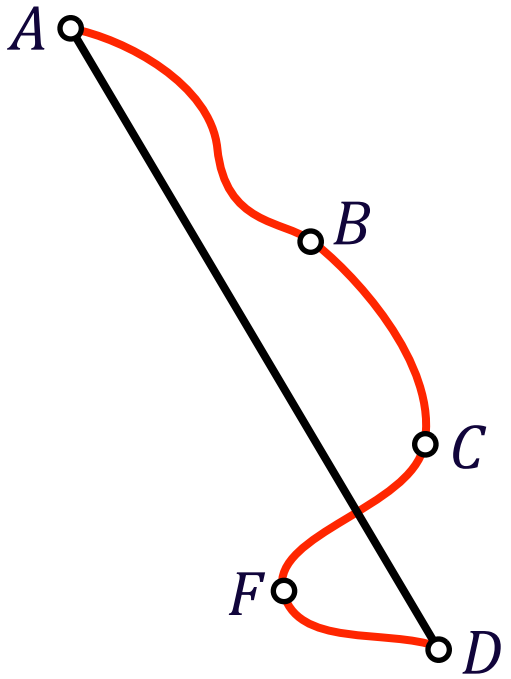
- Unguided, the search spreads out equally in all directions.
- Can we favour roads that go in roughly the right direction?

[Demo]

Estimating the distance still to go

We need to know if a road takes us towards or away from the destination D.

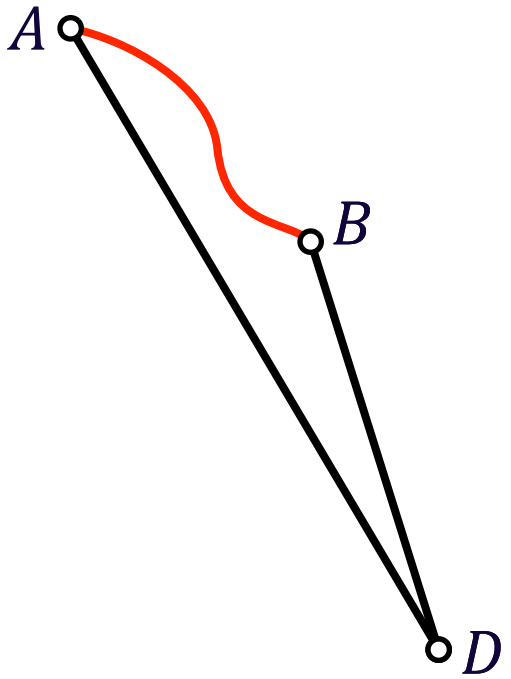
The straight-line distance $Dist(A, D)$ to the destination is a useful *underestimate* of the distance still to go.



Adjusting the road lengths

Let's modify the length of a road from A to B, reducing it by the amount it takes us closer to the destination.

$$Len'(A, B) = Len(A, B) - Dist(A, D) + Dist(B, D).$$



The lengths are still sensible

The length of each road stays positive:

$$Dist(A, D) \leq Dist(A, B) + Dist(B, D).$$

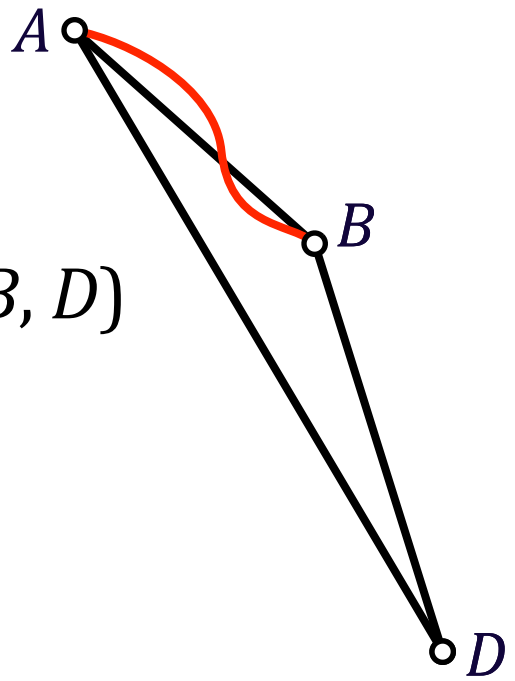
So $Len(A, B) \geq Dist(A, B)$

$$\geq Dist(A, D) - Dist(B, D)$$

And $Len'(A, B)$

$$= Len(A, B) - Dist(A, D) + Dist(B, D)$$

$$\geq 0.$$

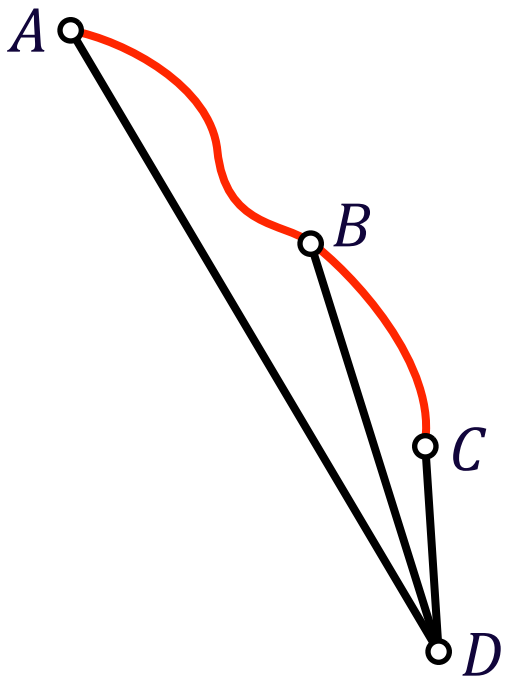


It all adds up

The adjustments add up nicely along paths:

$$Len'(A, B) + Len'(B, C)$$
$$= Len(A, B) + Len(B, C)$$
$$- Dist(A, D) + Dist(C, D).$$

That means any path from A to D will be adjusted by exactly $Dist(A, D)$.



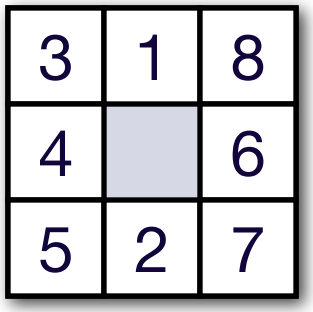
The result

- The algorithm still finds the shortest path to each place.
- But it prefers places that are closer to the destination.

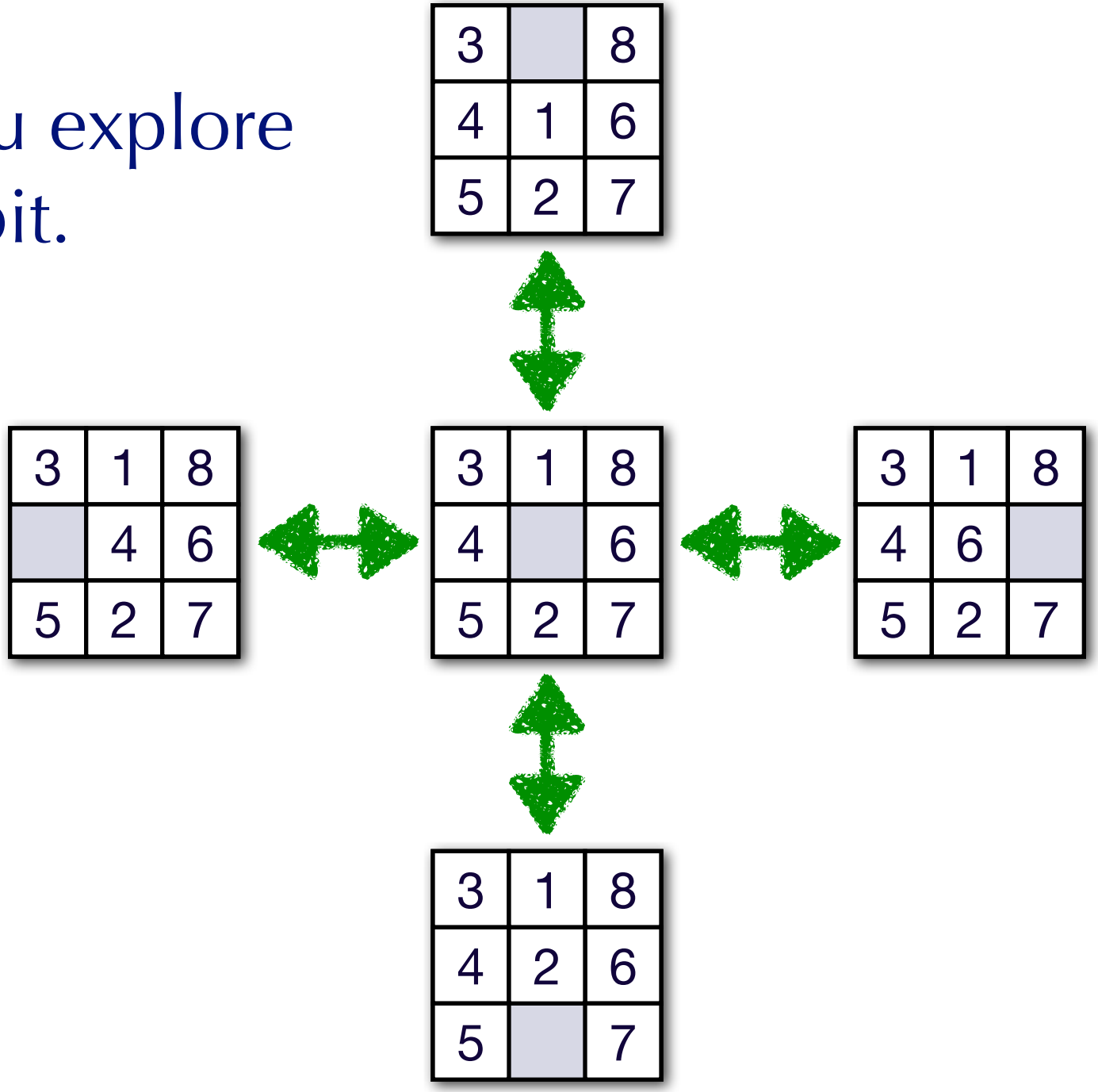
Faster, but still correct.

Another kind of map

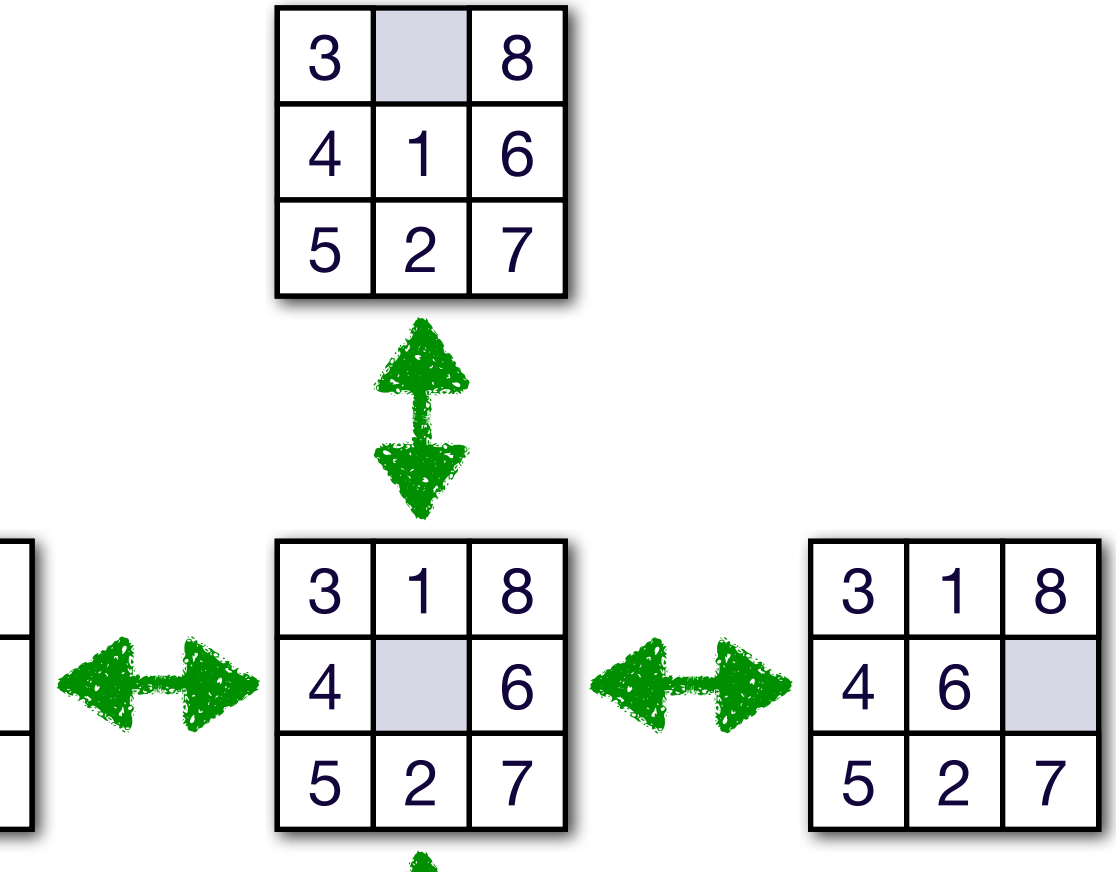
You wake up in an underground maze and see this sign:



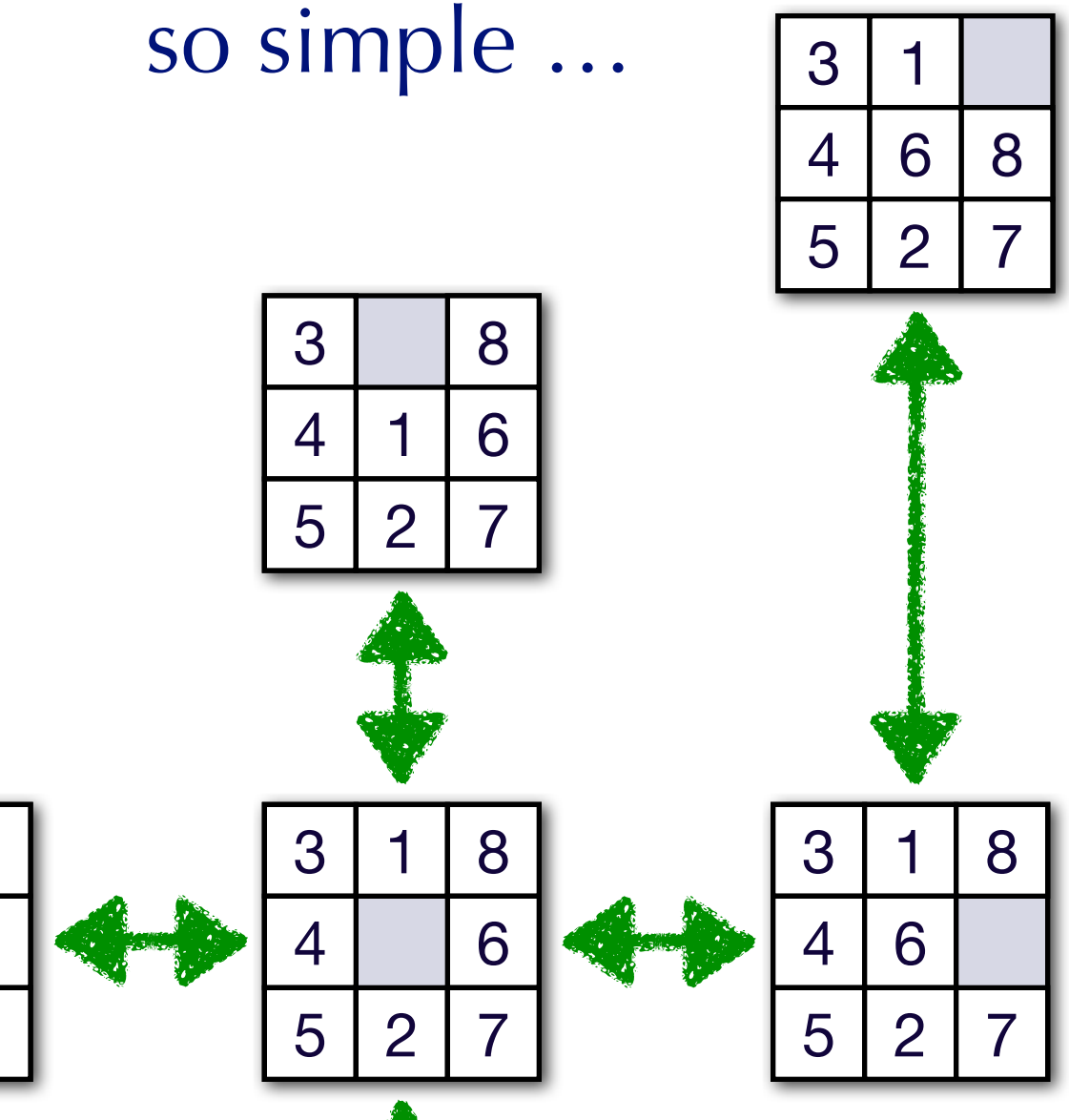
You explore a bit.



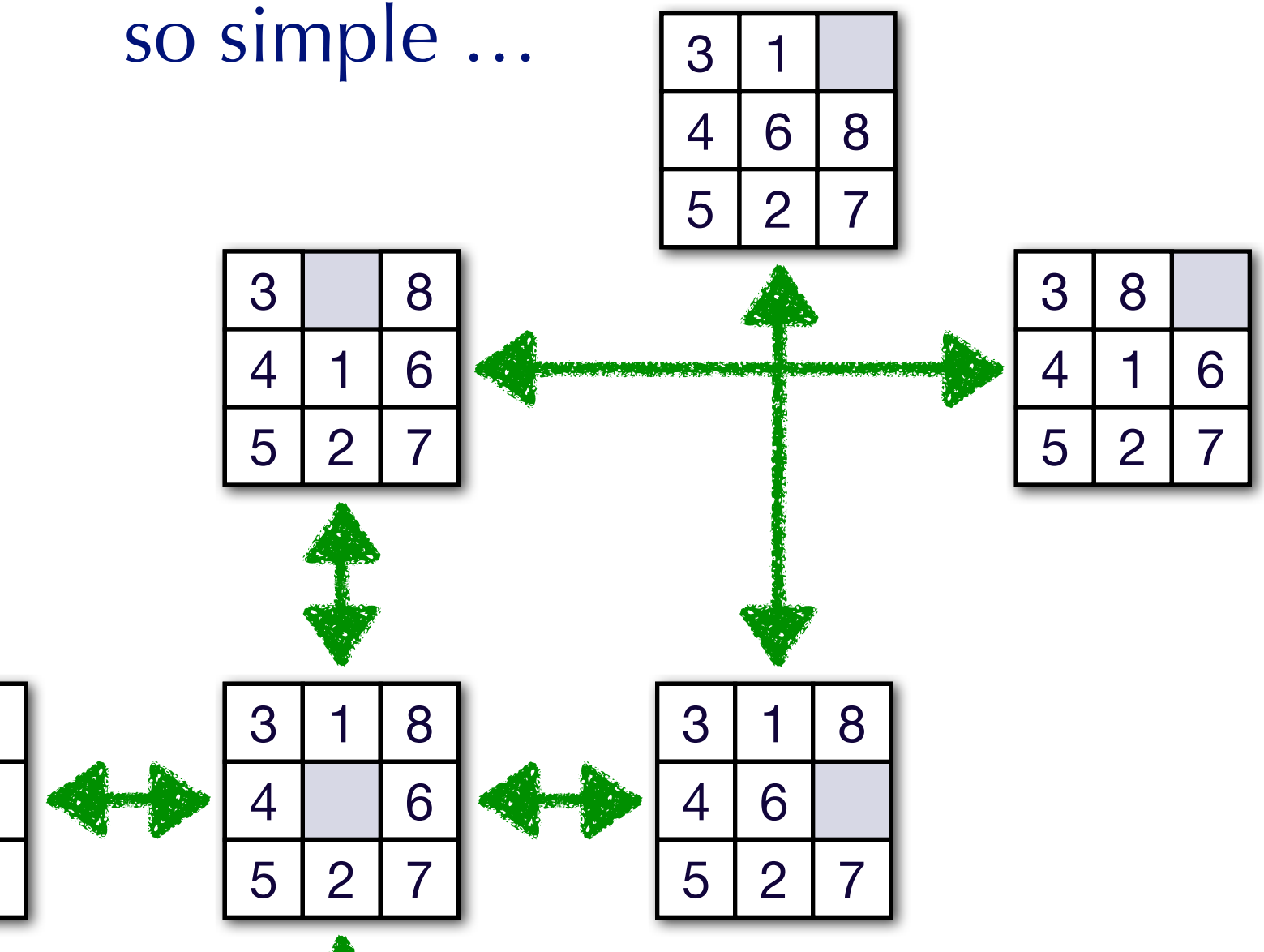
Things are not so simple …



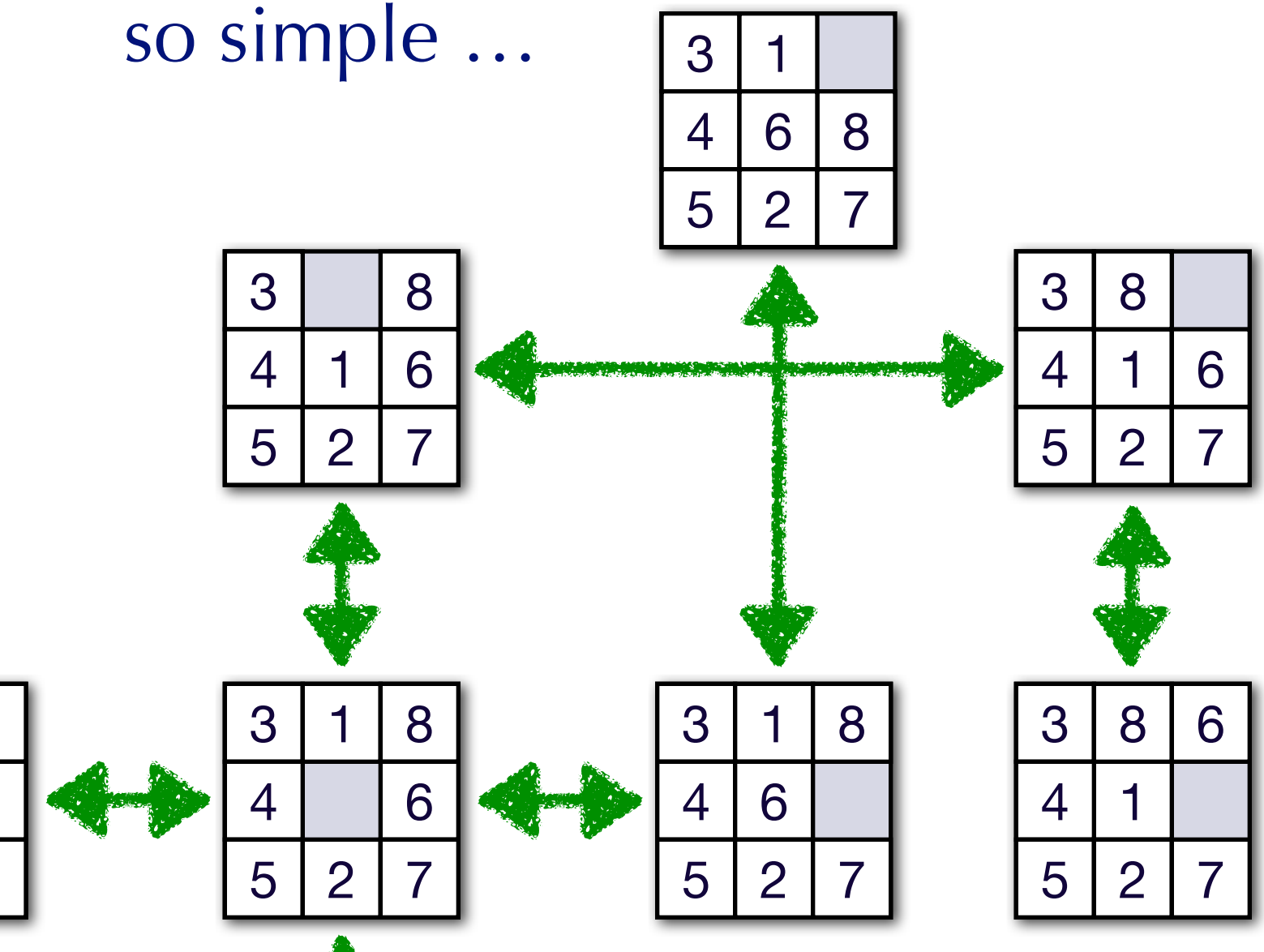
Things are not so simple …



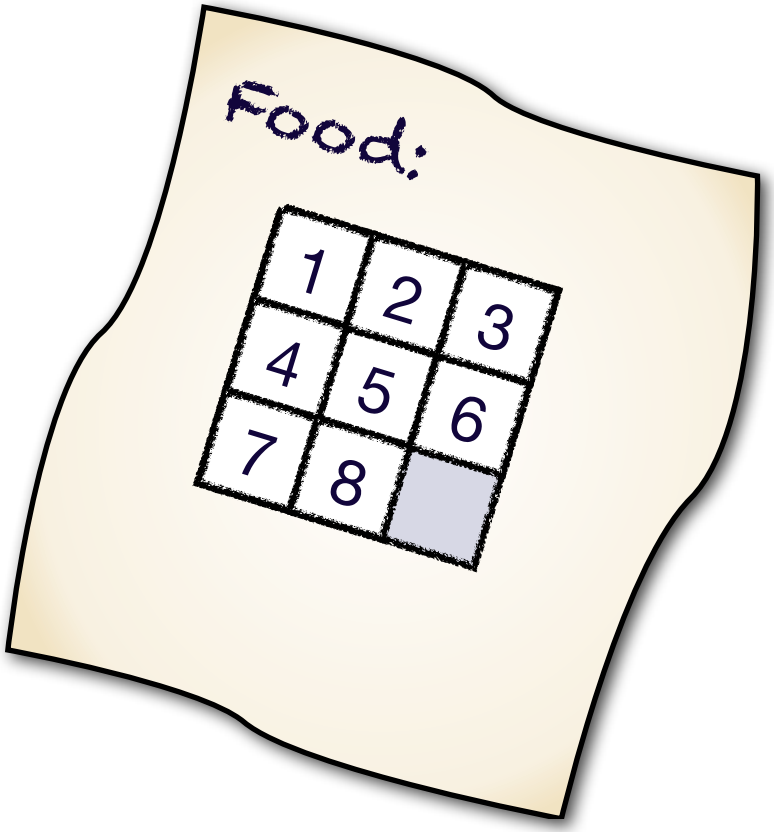
Things are not so simple …



Things are not so simple …



Just as you start to feel hungry …



Navigating the maze

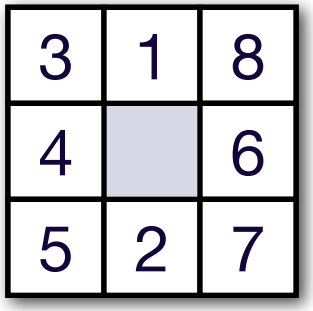
- Finding your way in the maze means solving the sliding block puzzle.
- Equally, we can solve the block puzzle by finding paths in the maze.

But the maze has about 9!/2 ≈ 180,000 junctions: we'd better be careful!

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

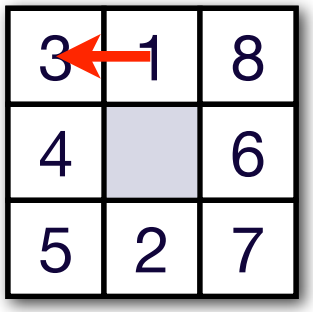
- The real number of moves can only be higher than this.



Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

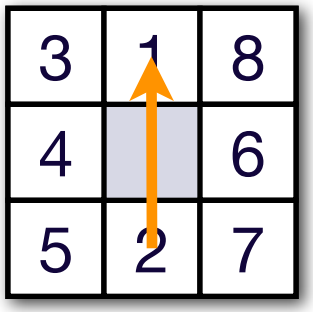


1

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

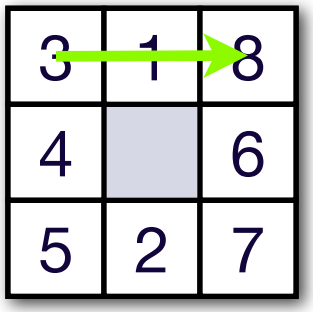


1+2

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

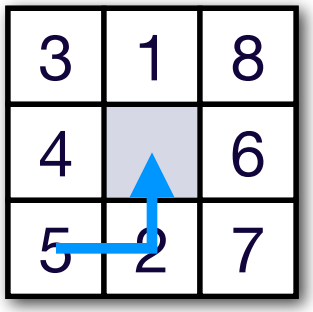


1+2+2

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

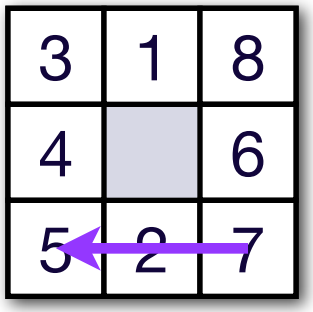


1+2+2+2

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

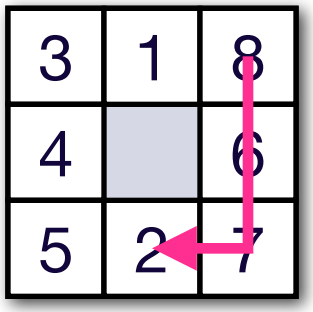


1+2+2+2+2

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.

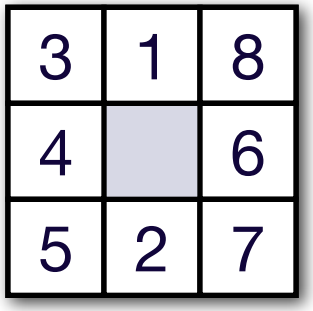


1+2+2+2+2+3

Estimating how far still to go

Add up the number of moves for each block *assuming they can slide over each other.*

- The real number of moves can only be higher than this.



1+2+2+2+2+3 = 12

Solving the puzzle by computer

- Explore the maze using Dijkstra's algorithm.
- Use the Manhattan distance as a heuristic.
- Create parts of the maze as needed.

[Demo]

Mathematics in Computer Science

- Does the program always give a correct answer?
- How long does the program take?
- Can the same problem be solved more quickly?

All these questions have mathematical answers.

Looking wider

- How can we be sure that information stays secure?
- How can we make many computers work together on a network?
- How can we be sure programs work the same way on different computers?

Mathematics helps with these questions too.